

Ferramenta de Comissionamento Digital Automatizado para Sistemas de Gestão de Edifícios

RELATÓRIO DE ESTÁGIO DE MESTRADO

Tiago Henrique Pereira Abreu

MESTRADO EM ENGENHARIA ELETROTÉCNICA - TELECOMUNICAÇÕES



UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

junho | 2026

Ferramenta de Comissionamento Digital Automatizado para Sistemas de Gestão de Edifícios

RELATÓRIO DE ESTÁGIO DE MESTRADO

Tiago Henrique Pereira Abreu

MESTRADO EM ENGENHARIA ELETROTÉCNICA - TELECOMUNICAÇÕES

ORIENTAÇÃO

Tiago Hipkin Meireles

COORIENTAÇÃO

Daniel Sousa

Índice

Índice.....	i
Índice de Figuras.....	iv
Índice de Tabelas.....	vii
Lista de Acrónimos.....	viii
Resumo.....	xi
Abstract.....	xiii
Agradecimentos.....	xv
1. Introdução.....	1
1.1. Motivação.....	1
1.2. Apresentação da empresa.....	2
1.3. Organização e conteúdos.....	3
2. Fundamentos Teóricos.....	5
2.1. Introdução ao <i>BMS (Building Management System)</i>	5
2.1.1. Arquitetura típica.....	6
2.2. Processo de comissionamento e normas aplicáveis.....	10
2.2.1. Etapas do comissionamento.....	11
2.2.2. Normas.....	12
2.3. Principais subsistemas de um BMS.....	14
2.3.1. Sistema AVAC (Aquecimento, Ventilação e Ar Condicionado).....	14
2.3.2. Sistema de iluminação.....	14
2.3.3. Sistema de gestão de energia.....	15
2.3.4. Sistemas de segurança.....	15
2.4. Protocolos usados em <i>BMS</i>	16
2.4.1. <i>Building Automation and Control Network (BACnet)</i>	17
2.4.2. Modbus.....	22
2.4.3. <i>KNX</i>	23
2.4.4. <i>Meter-bus (M-bus)</i>	25
2.4.5. <i>Digital Addressable Lighting Interface (DALI)</i>	26
2.5. Ferramentas digitais de apoio ao comissionamento.....	29
2.5.1. <i>WebCTRL</i>	29
2.5.2. <i>EcoStruxure Building Commission</i>	30
2.5.3. <i>AutoCXI</i>	31

2.5.4.	Limitações e oportunidades de inovação	31
2.6.	Automatização dos testes e qualidade dos dados.....	32
2.6.1.	A automatização no comissionamento	32
2.6.2.	Métodos e tecnologias de automatização	33
2.6.3.	Qualidade dos dados	34
3.	Ferramentas de automatização do planeamento e das configurações	37
3.1.	Ferramenta de Importação para o <i>BMS</i>	37
3.1.1.	Planeamento	37
3.1.2.	Implementação.....	40
3.2.	Ferramenta de comparação automática de versões	47
4.	Ferramenta de Testes	51
4.1.	Bibliotecas <i>BACnet</i>	51
4.2.	Planeamento	53
4.3.	Implementação	54
4.3.1.	Testes nas entradas analógicas.....	58
4.3.2.	Teste nas entradas digitais.....	60
4.3.3.	Teste nas saídas digitais.....	62
4.3.4.	Testes nas saídas analógicas	65
4.3.5.	Teste de pontos <i>multistate</i>	66
4.3.6.	Testes Sequenciais	66
4.3.7.	Monitorização	69
4.4.	Criação automática de relatórios para os testes.....	73
4.5.	Adaptações para o ambiente <i>Linux</i>	74
4.6.	Testes intermédios e finais	76
4.6.1.	Teste intermédio.....	76
4.6.2.	Testes finais e melhorias implementadas.....	78
5.	Conclusões.....	85
5.1.	Trabalhos futuros.....	87
	Referências.....	89
	Anexo A – Fluxograma da ferramenta de criação dos ficheiros <i>XML</i>	93
	Anexo B – Dicionários de campos extra	94
	Anexo B.1. – Pontos Físicos (Entradas/Saídas)	94
	Anexo B.2. – Pontos <i>BACnet</i>	95
	Anexo C – Código de comparação de listas de pontos	99

Anexo D – Código de procura de equipamentos e leitura de parâmetros <i>BACnet</i> ..	101
Anexo E – Tabela de filtros para o Teste de Limites	103
Anexo F – Fluxograma do funcionamento da ferramenta de testes	104
Anexo G – Carregamento da lista de pontos na ferramenta de testes	105
Anexo H – Página de testes <i>BACnet</i>	106
Anexo I – Código para a escrita de referências.....	107
Anexo I.1. – Nova versão corrigida	107
Anexo I.2. – Versão antiga.....	108

Índice de Figuras

Figura 1. Logótipo da empresa Blin Control [1].	2
Figura 2. <i>Software</i> do Sistema de Gestão de Edifícios usado pela empresa Blin Control.	2
Figura 3. Arquitetura de três níveis de um sistema <i>BMS</i> e protocolos dominantes em cada nível [5].	6
Figura 4. Exemplo de uma planta gráfica [7].	7
Figura 5. Exemplo de um diagrama de princípio [7].	7
Figura 6. Exemplo de um <i>room graphic</i> [7].	8
Figura 7. Fases do processo de comissionamento.	11
Figura 8. Integração dos protocolos na arquitetura de três camadas de um <i>BMS</i> [3].	16
Figura 9. Objeto <i>Analog Input</i> [16].	17
Figura 10. Objetos <i>BACnet</i> padrão [16].	18
Figura 11. Propriedades de um objeto <i>Analog Input</i> [16].	18
Figura 12. Propriedades de um objeto <i>BACnet</i> vistas no <i>YABE</i> .	20
Figura 13. Estrutura base de funcionamento do protocolo <i>KNX</i> [20].	24
Figura 14. Exemplo de um interruptor com suporte a <i>KNX</i> [20].	24
Figura 15. Arquitetura base de um sistema de iluminação <i>DALI</i> [23].	27
Figura 16. Exemplo de ficheiro de exportação de um equipamento.	39
Figura 17. Fluxo de Funcionamento da criação dos ficheiros para importação.	41
Figura 18. Ferramenta de criação dos ficheiros <i>XML</i> para importação no <i>EBO</i> .	41
Figura 19. Criação do cabeçalho <i>XML</i> .	42
Figura 20. Criação inicial de uma entrada ou saída de um equipamento.	43
Figura 21. Exemplo de conversão: a) Configuração no ficheiro Excel. b) Resultado da importação.	43
Figura 22. Exemplo de configuração de pontos físicos.	43
Figura 23. Preenchimento dos campos relativos ao ponto <i>BACnet</i> associado ao ponto físico.	44
Figura 24. Preenchimentos dos campos relativos ao ponto de <i>enable</i> .	44
Figura 25. Resultado da importação do ficheiro relativo ao <i>BACnet</i> .	44
Figura 26. Configuração dos pontos de <i>software</i> .	44
Figura 27. Importação dos pontos de <i>software</i> .	44
Figura 28. Exemplo de tela gráfica.	45
Figura 29. Estrutura de dados para a criação de telas gráficas.	45
Figura 30. Topo da função que cria a tabela de cada equipamento em <i>TGML</i> .	46

Figura 31. Interface gráfica do <i>software</i> de comparação de versões.	47
Figura 32. Estruturas de dados para comparação de ficheiros: a) classe de equipamentos b) classe de pontos.	48
Figura 33. Código para comparação de equipamentos.	48
Figura 34. Código inicial de comparação das listas de pontos.	48
Figura 35. Relatório de comparação de versões.	49
Figura 36. Cabeçalho do ficheiro de configuração dos testes.	53
Figura 37. Exemplo de configuração do ficheiro de equipamentos para testes.	53
Figura 38. Fluxo de funcionamento da ferramenta de testes funcionais.	54
Figura 39. Interface das configurações <i>BACnet</i>	55
Figura 40. Configuração da interface <i>BACnet</i>	56
Figura 41. Estrutura de dados para guardar a informação dos pontos na ferramenta de testes.	56
Figura 42. Fluxograma do processo de descoberta de equipamentos <i>BACnet</i> na rede.	57
Figura 43 Validação dos equipamentos e pontos <i>BACnet</i>	58
Figura 44. Verificação de limites num ponto analógico de entrada.	59
Figura 45. Exemplo do resultado de um Teste de Limites.	59
Figura 46. Janela de teste das entradas digitais.	60
Figura 47. Código para a escolha de um ponto da lista de pontos.	60
Figura 48. Código para leitura e aprovação de entradas digitais.	61
Figura 49. Exemplo de resultado de um teste de entradas digitais.	61
Figura 50. Aviso de segurança dos testes das saídas.	62
Figura 51. Janela de testes das saídas digitais.	62
Figura 52. Escrita de valores nos pontos digitais: a) configuração da interface gráfica b) função para escrita.	63
Figura 53. Código para validação de um ponto.	63
Figura 54. Fluxograma da limpeza de prioridades.	64
Figura 55. Código para garantir a escrita de um valor numérico.	65
Figura 56. Cabeçalho da janela de Testes Sequenciais.	67
Figura 57. Exemplo de validação manual de um Teste Sequencial do tipo Limites.	67
Figura 58. Código de escrita no teste sequencial.	68
Figura 59. Implementação para leitura e verificação de resultados do teste de Validação.	68
Figura 60. Implementação para o Teste de Limites.	69
Figura 61. Janela de monitorização.	70

Figura 62. Implementação para registo contínuo de valores.	70
Figura 63. Funções para controlar o tempo de monitorização.	71
Figura 64. Função para armazenar valores lidos durante a monitorização.	72
Figura 65. Exemplo de ficheiro de monitorização.	72
Figura 66. Gráfico de amostragem de um ponto <i>BACnet</i>	72
Figura 67. Exemplo de resumo de um teste de presença.	73
Figura 68. Exemplo de detalhes do teste de presença num equipamento.	73
Figura 69. Função para cálculo do resumo de um relatório.	74
Figura 70. Código para criar a tabela com informação dos testes dos pontos de cada equipamento.	74
Figura 71. Caixa de calha <i>DIN</i> para <i>Raspberry Pi</i> [41].	75
Figura 72. Adaptação da interface gráfica para ambiente <i>Linux</i>	76
Figura 73. Quadro de automação do Hotel <i>The Views Oásis</i>	77
Figura 74. Lista de equipamentos do Hotel <i>The Views Oásis</i> no <i>software YABE</i>	77
Figura 75. Resultados dos testes no Hotel <i>The Views Oásis</i>	78
Figura 76. Implementação para permitir a identificação de pontos diferentes com o mesmo identificador.	78
Figura 77. Importação de pontos <i>BACnet</i> num quadro de automação.	79
Figura 78. Esquema de ligação remota aos equipamentos dos hotéis.	79
Figura 79. Problema identificado com as unidades nos pontos <i>BACnet</i>	80
Figura 80. Código para adicionar a unidade a um ponto <i>BACnet</i>	80
Figura 81. Ficheiro <i>XML</i> com a criação de um ponto <i>BACnet</i> com indicação da unidade.	80
Figura 82. Janela de <i>bindings</i> de um ponto físico de entrada.	81
Figura 83. Janela de <i>bindings</i> de um ponto físico de saída.	81
Figura 84. Exemplo de lógica de controlo.	82
Figura 85. Relatório de um teste sequencial a um ventilador.	82

Índice de Tabelas

Tabela 1. Aplicações de cada prioridade do protocolo <i>BACnet</i> [18].....	21
Tabela 2. Comparação das ferramentas de comissionamento.	32
Tabela 3. Codificação das unidades no <i>EBO</i>	39

Lista de Acrónimos

AES	<i>Advanced Encryption Standard</i>
AHU	<i>Air Handler Units</i>
ASC	<i>Application Specific Controllers</i>
ASCII	<i>American Standard Code for Information Interchange</i>
ASHRAE	<i>American Society of Heating Refrigerating and Air-Conditioning Engineers</i>
AVAC	<i>Aquecimento Ventilação e Ar Condicionado</i>
BACnet	<i>Building Automation and Control Network</i>
BACnet/SC	<i>BACnet Secure Connect</i>
BBMD	<i>BACnet Broadcast Management Device</i>
BMS	<i>Building Management System</i>
BSRIA	<i>Building Services Research and Information Association</i>
CAV	<i>Constant Air Volume</i>
CIBSE	<i>Chartered Institution of Building Services Engineers</i>
COM	<i>Component Object Model</i>
DA	<i>Digital Addressable</i>
DALI	<i>Digital Addressable Lighting Interface</i>
DiiA	<i>Digital Illumination Interface Alliance</i>
EBO	<i>EcoStruxure Building Operation</i>
EIB	<i>European Installation Bus</i>
FPC	<i>Free-programmable Controllers</i>
FPT	<i>Functional Performance Testing</i>
HTTPS	<i>Hypertext Transfer Protocol Secure</i>
HVAC	<i>Heating Ventilation and Air Conditioning</i>
IPMVP	<i>International Performance Measurement and Verification Protocol</i>
IT-WG	<i>Eurosai IT Working Group</i>
LED	<i>Light Emitting Diode</i>
MS/TP	<i>Master-Slave/Token-Passing</i>
NIBS	<i>National Institute of Building Sciences</i>
NTC	<i>Negative Temperature Coefficient</i>
OMS	<i>Open Metering System</i>
OPR	<i>Owner's Project Requirements</i>
PLC	<i>Programmable Logic Controller</i>
PSU	<i>Power Supply Unit</i>
PTC	<i>Positive Temperature Coefficient</i>
PTP	<i>Point-To-Point</i>
RTU	<i>RoofTop Units</i>
SELV	<i>Safety Extra Low Voltage</i>
TBCxP	<i>Total Building Commissioning Process</i>
TCP	<i>Transmission Control Protocol</i>
TGML	<i>Technical Graphics Markup Language</i>
TLS	<i>Transport Layer Security</i>
VAV	<i>Variable Air Volume</i>
VMAC	<i>VAV Modular Actuator Controllers</i>
XML	<i>Extensible Markup Language</i>

YABE *Yet Another BACnet Explorer*

Resumo

O comissionamento de Sistemas de Gestão de Edifícios (*Building Management Systems - BMS*) desempenha um papel fundamental na garantia do desempenho, segurança e eficiência operacional das infraestruturas modernas. No entanto, este processo é tradicionalmente realizado de forma manual e repetitiva, o que implica um elevado consumo de tempo e uma elevada suscetibilidade ao erro humano, especialmente nas instalações de grande dimensão. O presente trabalho descreve o desenvolvimento e a validação de um conjunto de ferramentas de *software*, concebidas em linguagem *Python*, focadas na automatização e digitalização das etapas de configuração e testes funcionais do comissionamento. A solução implementada assenta em três ferramentas modulares e versáteis: uma ferramenta de importação, que automatiza a criação de ficheiros *XML* e *TGML* para o *software EcoStruxure Building Operation* a partir de um ficheiro de planeamento em *Excel*; uma ferramenta de comparação de versões, destinada a identificar rapidamente alterações entre revisões do projeto e assegurar a integridade dos dados; e uma ferramenta de testes, que estabelece comunicação direta com equipamentos *BACnet/IP*. Esta última permite a execução de testes de limites, validações de entradas e saídas dos equipamentos, testes sequenciais e monitorização de dados em tempo real, permitindo criar, de forma automática, relatórios de conformidade em formato *PDF*.

Adicionalmente, foi garantida a portabilidade da aplicação de testes para ambiente *Linux*, com o objetivo de permitir a instalação permanente nos quadros de automação de um equipamento pequeno e de baixo consumo para diagnósticos de forma remota.

A validação prática em ambiente real, realizada nos hotéis *The Views Oásis*, *Alto Lido* e *Baía Azul*, demonstrou a eficiência da ferramenta de testes na redução do tempo de execução das tarefas relacionadas com a manutenção e na deteção de falhas. Esta validação também permitiu identificar alguns problemas com a primeira ferramenta desenvolvida, que foram posteriormente resolvidos, e implementar melhorias.

Conclui-se que o sistema desenvolvido constitui uma solução eficiente e de baixo custo para a modernização do setor, promovendo a repetibilidade dos testes e um padrão de qualidade da documentação criada.

Palavras-chave: *BMS*, Comissionamento Digital, Automatização, *BACnet*, *Python*.

Abstract

The commissioning of Building Management Systems plays a fundamental role in ensuring the performance, safety, and operational efficiency of modern infrastructures. However, this process is traditionally carried out manually and repetitively, resulting in significant time consumption and a high susceptibility to human error, particularly in large-scale installations. This work describes the development and validation of a set of software tools, developed in Python, focused on the automation and digitalization of configuration and functional testing stages during the commissioning process. The implemented solution is based on three modular and versatile tools: an import tool, responsible for automating the creation of XML and TGML files for the EcoStruxure Building Operation software from an Excel planning file; a version comparison tool, designed to quickly identify changes between project revisions and ensure data integrity; and a testing tool, which enables direct communication with BACnet/IP devices. The latter enables the execution of limit validation tests, validation of device inputs and outputs, sequential tests, and real-time data monitoring, while also automatically generating PDF compliance reports.

Additionally, the testing application was adapted to operate in a Linux environment, enabling its permanent installation in automation panels using a small, low-power device for remote diagnostics.

Practical validation in real environments, carried out at the The Views Oásis, Alto Lido, and Baía Azul hotels, demonstrated the effectiveness of the testing tool in reducing the execution time of maintenance-related tasks and in detecting faults. This validation process also made it possible to identify issues in the first developed tool and implement further improvements.

The developed system represents an efficient and low-cost solution for modernizing the sector, promoting test repeatability and improving the quality standard of the generated documentation.

Keywords: BMS, Digital Commissioning, Automation, BACnet, Python.

Agradecimentos

Com a conclusão deste trabalho, encerro uma etapa fundamental do meu percurso académico e pessoal. Este projeto não teria sido possível sem o contributo e o apoio de diversas pessoas e entidades, às quais gostaria de expressar o meu sincero reconhecimento.

Ao meu orientador, Professor Tiago Meireles, agradeço por todo o apoio e pela disponibilidade demonstrada ao longo deste trabalho. A sua orientação crítica, o acompanhamento constante, e o apoio na estruturação e revisão deste relatório foram fundamentais para garantir o rigor académico necessário a este trabalho.

À empresa Blin Control, pela oportunidade concedida para a realização deste estágio e por me terem acolhido de forma exemplar. Um agradecimento especial ao Eng.º Daniel Sousa e Eng.º Jorge Neves, pela disponibilidade e por me terem proporcionado os meios técnicos e o ambiente de trabalho necessários para o desenvolvimento das ferramentas aqui apresentadas.

Aos meus amigos e colegas de curso, agradeço por todos os momentos partilhados ao longo destes anos. O apoio mútuo, a troca de ideias e a amizade demonstrada nos momentos de maior exigência foram essenciais para tornar este percurso mais enriquecedor e gratificante.

Aos meus pais e restante família, por serem o meu suporte incondicional. Agradeço todo o esforço, dedicação e os meios que me proporcionaram para que esta etapa fosse alcançável, bem como pela motivação constante que sempre me transmitiram ao longo de toda a minha vida académica.

Por último, mas de forma muito especial e *in memoriam*, gostaria de prestar uma sentida homenagem aos meus avós. O seu exemplo de vida e os valores que me transmitiram continuam a ser uma fonte de motivação constante em cada etapa superada. Este trabalho, e o ciclo que agora encerro, são também um tributo à sua presença, que permanece indelével na minha vida.

1. Introdução

Ao longo deste estágio pretendeu-se desenvolver um conjunto de ferramentas de *software* capazes de automatizar o processo de comissionamento de sistemas *Building Management System (BMS)*, com o objetivo de reduzir o tempo necessário à execução deste processo e minimizar a ocorrência de erro humano. A solução implementada baseou-se na leitura de listas de pontos e na aquisição de dados em tempo real a partir dos equipamentos, o que permite a comparação entre valores esperados e valores efetivamente medidos, bem como a execução automática de testes simples de validação.

As ferramentas desenvolvidas foram concebidas para serem versáteis e modulares, para permitir a sua adaptação a diferentes contextos e tipos de instalações. No entanto, o seu desenvolvimento foi orientado para responder às necessidades específicas de uma empresa do setor, a Blin Control, tendo em consideração os formatos de dados utilizados, os protocolos de comunicação adotados, nomeadamente o *BACnet/IP* e a aplicação *BMS* implementada nos edifícios dos seus clientes.

Este trabalho consistiu num esforço de digitalização e modernização do processo de comissionamento, contribuindo para a melhoria da eficiência operacional, da qualidade do processo e da documentação gerada. Adicionalmente, a solução desenvolvida promove uma maior padronização e repetibilidade dos testes, facilitando futuras atividades de manutenção e diagnóstico ao longo do ciclo de vida dos edifícios.

1.1. Motivação

Um Sistema de Gestão de Edifícios desempenha um papel fundamental no que diz respeito à operação segura e eficiente das instalações, permitindo a monitorização e controlo de sistemas como AVAC (Aquecimento, Ventilação e Ar Condicionado), iluminação, energia e segurança. Para garantir que estes sistemas funcionem de acordo com o projeto e os requisitos operacionais, é indispensável a realização do comissionamento, processo inicial de planeamento e execução, mas que também valide a correta implementação, comunicação e funcionamento dos equipamentos e respetivos pontos.

No entanto, o comissionamento de sistemas *BMS* é maioritariamente realizado de forma manual, muitas vezes com a verificação individual de centenas de pontos através da documentação existente e da observação direta em campo. Este método apresenta diversas limitações, nomeadamente o elevado consumo de tempo, dependência da experiência dos técnicos, dificuldade para a repetição sistemática dos testes, fraca rastreabilidade dos resultados e elevada probabilidade de erro humano, sobretudo em instalações de grande dimensão.

Apesar do aumento na digitalização, existem ainda poucas ferramentas que integram de forma automática a leitura de listas de pontos, a aquisição de dados em tempo real através de protocolos como o *BACnet* e a validação dos resultados obtidos. Este tipo de solução permitiria padronizar o processo de comissionamento, a criação automática de relatórios e a reutilização dos testes já definidos ao longo do ciclo de vida do edifício. Deste modo, coloca-se o problema de automatizar o processo de

comissionamento de sistemas BMS, com os objetivos de reduzir o tempo de execução do comissionamento e o erro humano.

Este trabalho propõe o desenvolvimento de uma ferramenta de comissionamento digital automatizado, capaz de interpretar listas de pontos, comunicar com equipamentos em tempo real, executar testes básicos de verificação e criar relatórios de conformidade de forma sistemática.

1.2. Apresentação da empresa

A Blin Control, cujo logótipo encontra-se na figura 1, é uma empresa com sede no Centro Empresarial da Madeira, município do Funchal, sendo especialista na área da Gestão Técnica de Edifícios. Os seus técnicos e engenheiros têm a capacidade de integrar diversos sistemas técnicos, incluindo climatização, ventilação, iluminação e segurança, num único sistema de gestão, operação e controlo.



Figura 1. Logótipo da empresa Blin Control [1].

A empresa conta com uma grande presença no setor hoteleiro da Ilha da Madeira, sendo que também presta serviços para centros comerciais e outros grandes edifícios com o objetivo de otimizá-los de acordo com as necessidades dos seus clientes.

Faz parte da rede de parceiros certificados *EcoXpert* da *Schneider Electric*, sendo reconhecida como uma empresa especialista na área dos Sistemas de Automação de Edifícios. O principal *software* usado pela empresa encontra-se na figura 2. Este é o *EcoStruxure Building Operation* da *Schneider Electric* que permite fazer toda a gestão do sistema técnico de um edifício. A Blin Control intervém em todas as etapas de implementação dos sistemas de automação, desde a fase de conceção e projeto técnico até à execução da ligação física dos dispositivos e prestação de suporte técnico especializado.

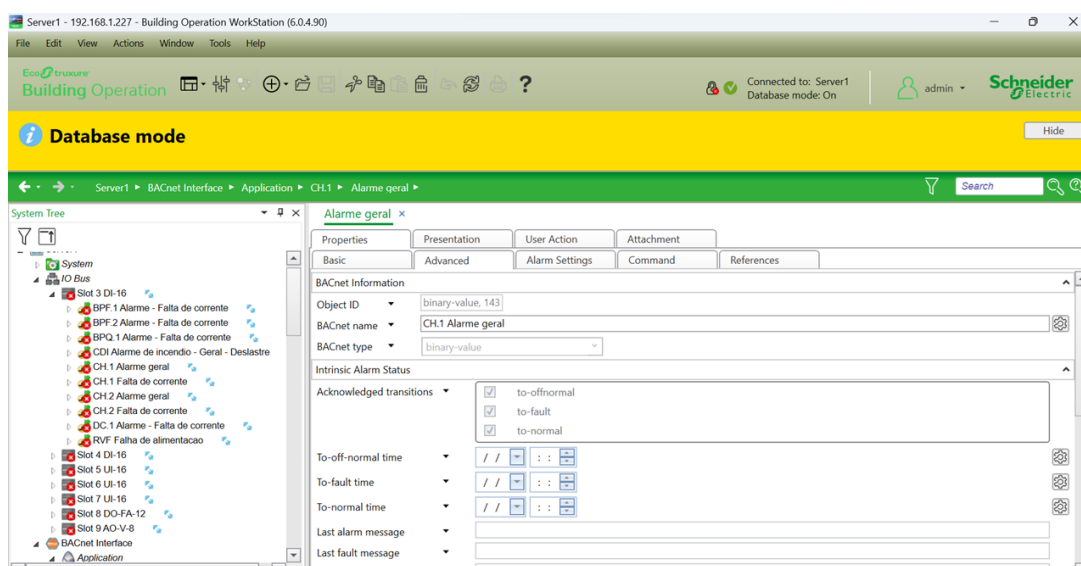


Figura 2. *Software* do Sistema de Gestão de Edifícios usado pela empresa Blin Control.

1.3. Organização e conteúdos

O presente relatório encontra-se estruturado em seis capítulos, estando organizado da seguinte forma:

No capítulo 1, Introdução, foi apresentado o enquadramento do trabalho, descreveu-se a motivação e os objetivos traçados para o estágio. É ainda incluída uma breve apresentação da empresa Blin Control, onde decorreu o estágio.

No capítulo 2, Estado da Arte, aborda-se a fundamentação teórica e técnica necessária para a compreensão dos temas abordados ao longo do desenvolvimento do trabalho. Inicia-se com uma introdução aos Sistemas de Gestão de Edifícios e respetivas arquiteturas, seguida de uma análise detalhada dos protocolos de comunicação mais comuns, com particular atenção ao protocolo *BACnet*. São também abordados os processos e normas do comissionamento, bem como as ferramentas digitais existentes e a importância da automatização de testes para a qualidade dos dados.

No capítulo 3, Ferramentas de automatização do planeamento e das configurações, descreve-se o desenvolvimento de uma aplicação em *Python* concebida para automatizar o processo de caracterização do Sistema de Gestão de Edifícios no *software EcoStruxure Building Operation* a partir de dados presentes num ficheiro de planeamento *Excel*. Este processo é feito a partir da criação de ficheiros *XML* e *TGML* que contêm a informação necessária do sistema e que podem ser importados no *software*. Apresenta-se, também, o desenvolvimento de uma ferramenta destinada a assegurar a integridade dos dados ao longo das revisões do projeto, permitindo a identificação rápida de diferenças entre diferentes versões de listas de pontos.

No capítulo 4, Ferramenta de Testes, aborda-se a implementação da ferramenta de testes em edifícios. São analisadas as bibliotecas *BACnet* existentes, a metodologia de execução de testes funcionais e sequenciais, e a monitorização de dados em tempo real. Adicionalmente, descrevem-se as adaptações realizadas para o ambiente *Linux* e os resultados práticos obtidos nos testes realizados em algumas instalações hoteleiras.

No capítulo 5, Conclusões, faz-se uma síntese das principais conclusões e contributos do trabalho realizado e também são apresentadas propostas para desenvolvimentos e trabalhos futuros.

2. Fundamentos Teóricos

Antes de iniciar qualquer tentativa de implementação prática, importa entender os conceitos fundamentais que fazem parte de um *Building Management System*. Neste capítulo abordar-se-ão estes conceitos, começando por uma introdução para que se perceba de um modo geral o que é um *BMS*, as suas principais funções e a arquitetura base do sistema. Posteriormente, será feita uma análise aos protocolos de comunicação mais usados e como permitem criar interoperabilidade entre todos os equipamentos.

Os dois temas referidos no parágrafo anterior formam a base de um *Building Management System*. Será abordado o processo de comissionamento, ponto fundamental deste trabalho, e serão analisadas as ferramentas digitais de apoio ao comissionamento existentes no mercado, quais as suas limitações e oportunidades de inovação.

2.1. Introdução ao *BMS* (*Building Management System*)

Um *BMS* é um sistema de controlo instalado em edifícios para gerir e monitorizar equipamentos de um edifício, bem como a sua parte elétrica, sistemas de ar condicionado, iluminação, equipamentos de segurança, sistemas de rega, controlo de ocupação, aquecimento de piscinas, defumagem, entre outros [2].

Este tipo de sistemas visa a eficiência operacional de um edifício e o conforto dos seus ocupantes, enquanto minimiza o consumo energético e os custos de operação. Ou seja, estes sistemas permitem que os edifícios funcionem de forma mais eficiente, segura e sustentável [2].

Podem ser integrados diversos sistemas como *HVAC* (*Heating, Ventilation and Air Conditioning*), iluminação, sistemas de segurança e sensores de qualidade do ar, numa interface centralizada. Esta integração permite aos gestores dos edifícios controlar os diferentes sistemas numa interface comum e centralizada, quer seja no ajuste do ar condicionado, gestão do controlo de acesso ou monitorização do consumo energético em tempo real. Os sistemas recolhem dados provenientes de sensores e outros equipamentos espalhados pelo edifício. Estes dados são analisados para otimizar a performance do edifício na diminuição do consumo energético e até na antecipação de manutenções em caso de necessidade [2].

Há vários benefícios de um *BMS* que levam as empresas públicas e privadas a procurar este tipo de solução [2]:

- Redução do consumo energético através da gestão eficiente dos sistemas *HVAC*, iluminação e outros sistemas de consumo energético significativo;
- Automação de tarefas recorrentes, permitindo que os gestores possam dedicar-se a tarefas mais críticas;
- Aumento no conforto e segurança;
- Auxílio nas decisões de operação e manutenção através da recolha e análise de dados;
- Controlo centralizado sobre todos os sistemas através de uma interface única, por vezes, com acesso por dispositivos móveis;

- Suporte à integração de fontes de energia renovável, o que ajuda a alcançar os objetivos de sustentabilidade energética.

2.1.1. Arquitetura típica

No início da implementação dos *BMS*, os sistemas eram feitos para serem autônomos e isolados. Posteriormente, a integração da gestão de edifícios com a rede *IP*, por meio de encapsulamento de protocolos antigos sobre os protocolos *Ethernet* e *IP*, permitiu reduzir custos e aumentar a conveniência [3].

Um *BMS* é projetado para aumentar a segurança, o conforto, diminuir custos e aumentar a sustentabilidade ambiental. De acordo com a norma *EN ISO 16484* [4], que especifica as fases necessárias para um projeto *BMS* e os equipamentos necessários para desempenhar corretamente as suas tarefas, existem três camadas distintas que organizam a estrutura de um projeto de um *BMS*: gestão, automação e campo, que permitem organizar toda a estrutura do sistema como demonstra a figura 3 [3]:

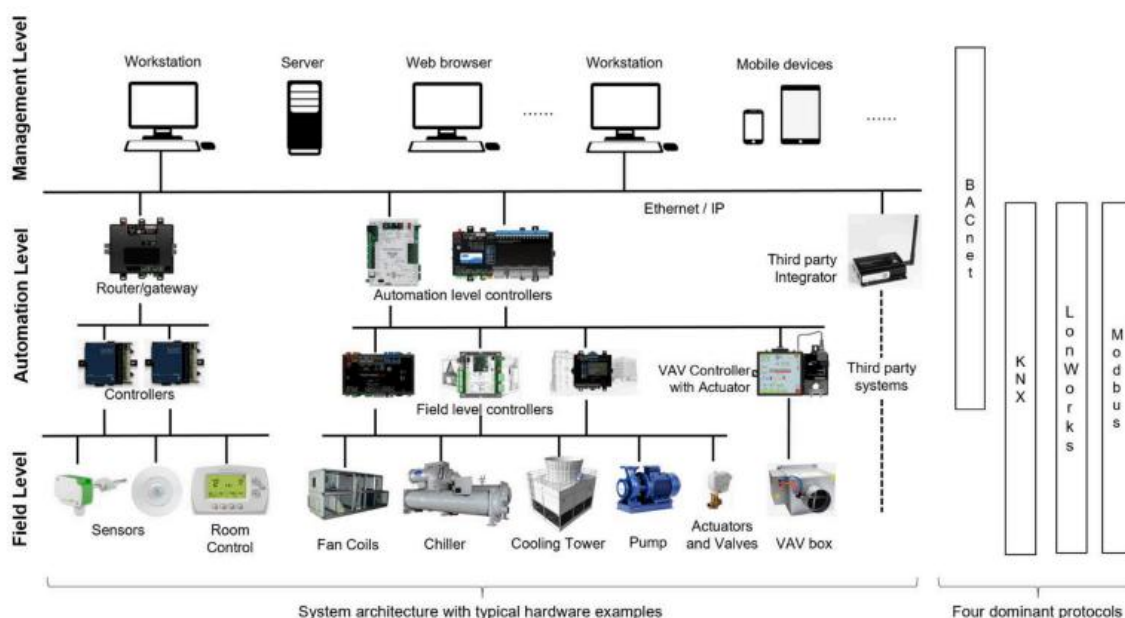


Figura 3. Arquitetura de três níveis de um sistema *BMS* e protocolos dominantes em cada nível [5].

A camada de gestão (*management level*) contém as estações de operação (*workstations*), monitorização e unidades de programação (*programming units*) que processam os dados e dão suporte à monitorização e administração do sistema de gestão [3]. De uma forma geral, contém a interface humana (estações de trabalho), servidores, dispositivos de *routing*, conectados por protocolos de comunicação como *TCP/IP* e *BACnet* [5].

Na **camada de gestão** é possível ter acesso a recursos que fornecem informações sobre o estado do sistema. O primeiro são as plantas gráficas (*floor plan graphics*), como ilustra a figura 4, que mostram a condição do edifício em relação ao plano do piso. Podem, por exemplo, mostrar um mapa de calor para ajudar a visualizar a diferença de temperatura em relação à temperatura pretendida [6].

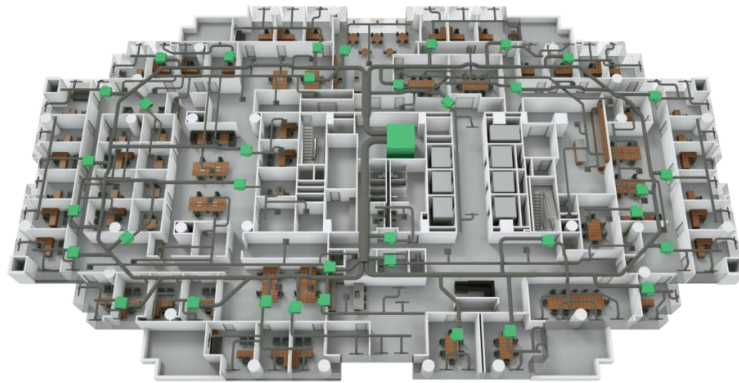


Figura 4. Exemplo de uma planta gráfica [7].

Na Figura 5 tem-se um exemplo de um diagrama de princípio (*system graphics*) que mostra de forma visual o estado de um sistema que desempenha uma determinada função. Mostra relações entre sensores, atuadores, controladores e equipamentos que fornecem dados para criar mapas com fluxos de energia (ar, água, eletricidade) e sinais de controlo. Servem como base para projeto, operação e manutenção, permitindo compreender rapidamente a arquitetura funcional do sistema. [6].

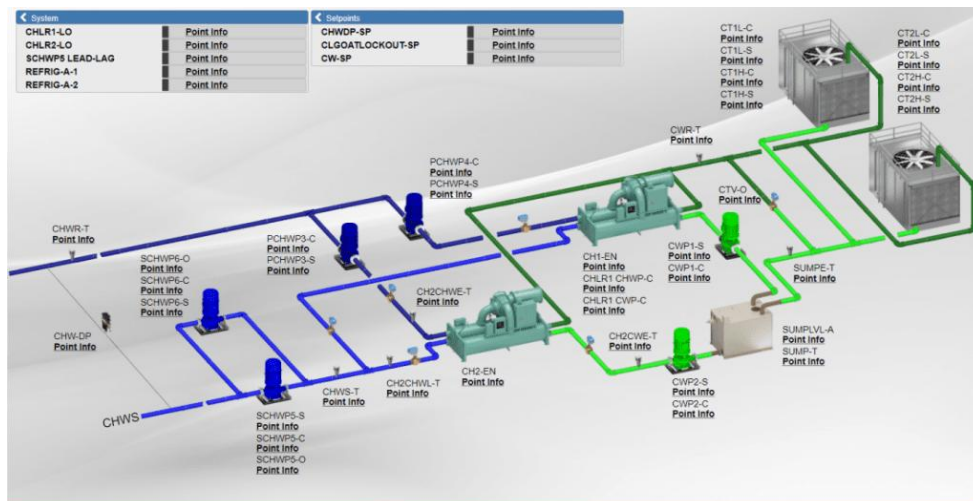


Figura 5. Exemplo de um diagrama de princípio [7].

Por fim, para visualizar uma representação de uma unidade terminal (*terminal unit*) são usados os *room graphics*. Eles permitem a exibição do estado do equipamento (ligado/desligado, caudal de ar, temperatura, potência, entre outros) e permitem ajustar parâmetros locais, como *setpoints* de temperatura e velocidade [6]. A figura 6 representa uma unidade de ar variável para fornecer ar condicionado a uma sala.

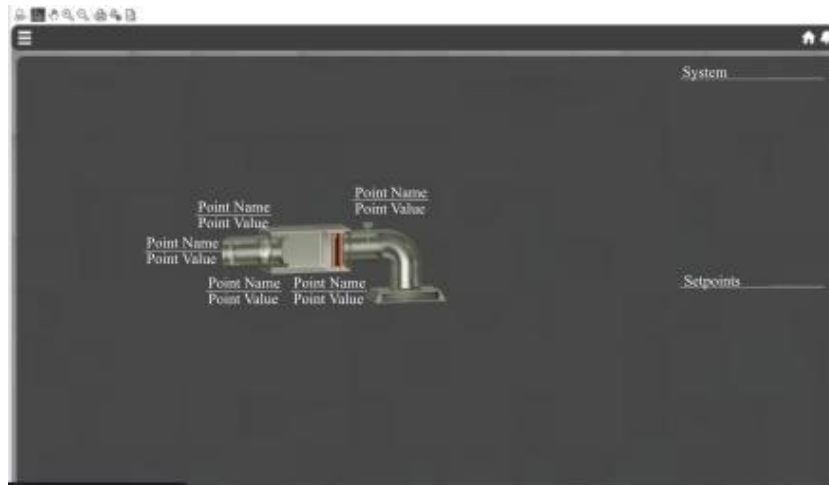


Figura 6. Exemplo de um *room graphic* [7].

A **camada de automação** representa, normalmente, uma rede dedicada de comunicação usada para interligar dispositivos que têm como objetivo principal controlar e automatizar o edifício. Esta camada inclui os controladores globais do edifício como sistemas de produção de energia, *air handling units* e sistemas de refrigeração (*chillers*). Ou seja, contém todos os controladores da planta principal do edifício. Contém ainda os principais dispositivos de controlo e funções de automação intermédias das infraestruturas técnicas do edifício. Integra controladores em rede que asseguram a execução local de algoritmos de controlo e a coordenação entre equipamentos. O funcionamento é suportado por protocolos normalizados de comunicação, como o *BACnet* e *KNX*, que garantem a interoperabilidade e a troca estruturada de informação entre os diferentes subsistemas [3] [5].

Na camada de automação encontram-se os controladores de campo (*field controllers*) que utilizam dados de entrada e saída para monitorizar e controlar outros dispositivos. Alguns destes dispositivos conseguem operar de forma independente em caso de falha no sistema de supervisão (*supervisory device*). As entradas são a forma como o controlador adquire a informação proveniente dos dispositivos de controlo (interruptores de estado, sensores de temperatura, pressão, humidade, entre outros). As saídas são usadas para controlar dispositivos (essencialmente atuadores e relés). Podem ser usadas variáveis de rede (*network variables*) para receber dados de entrada de outros controladores e enviar dados de saída para outros controladores. Os controladores de campo são divididos, essencialmente em três categorias [6]:

- *Free-programmable controllers (FPC)*: controladores totalmente programáveis, em que a lógica tem de ser criada do zero pelo programador. Isto permite ter elevada flexibilidade para edifícios complexos ou com necessidades especiais.
- *Application specific controllers (ASC)*: controladores com apenas um propósito. Podem permitir alguma customização através de um menu, mas não podem ser programados livremente.
- *VAV (variable air volume) modular actuator controllers (VMAC)*: controladores híbridos usados em unidades de volume de ar variável. À primeira vista, poderiam ser interpretados como um ASC, mas são controladores totalmente programáveis.

A camada de campo compreende os sensores, atuadores e demais dispositivos diretamente associados aos equipamentos e sistemas técnicos do edifício. Estes

elementos constituem unidades físicas autónomas, podendo ser controladores de aplicação específica ou controladores genéricos. No caso dos controladores dedicados a aplicações específicas, o seu funcionamento baseia-se em protocolos de comunicação como o *Modbus*, o *KNX* ou outros protocolos proprietários definidos pelo fabricante [5].

É na **camada de campo** que estão os atuadores: dispositivos mecânicos que usam um motor para movimentar uma válvula ou registo de ar (*damper*). O atuador recebe o sinal do controlador de campo e com base nesse sinal atua sobre o motor que, por sua vez, é movimentado até à posição desejada. Existem três formas de controlo [6]:

- *On/off control*: existem essencialmente dois estados, ligado ou desligado. É enviada energia ao atuador para a sua abertura e é retirada a energia para voltar a fechar. Tem um comportamento semelhante a um relé.
- *Floating control*: permite ao atuador movimentar o motor em sentido horário ou anti-horário, através de um controlo por um fio comum e dois fios de alimentação para controlar a rotação.
- Controlador proporcional: usa um sinal de controlo para indicar uma posição ao atuador.

Todos os atuadores devem ser projetados com uma posição de segurança (*fail-to-position*). Ou seja, a posição do atuador, caso a energia seja removida. Geralmente são usadas molas para garantir uma posição aberta ou fechada [6].

Outro equipamento da camada de campo são os sensores. Os sensores mais comuns podem ser separados em três categorias: sensores resistivos, sensores alimentados e sensores de circuito ou comutação [6].

Os sensores resistivos utilizam a variação da resistência para medir a grandeza pretendida, como a temperatura. Neste caso, existem os *Positive Temperature Coefficient (PTC)* e os *Negative Temperature Coefficient (NTC)* em que a resistência varia de forma direta ou inversa com a temperatura, respetivamente. Neste tipo de sensores a calibração é fundamental [6].

Os sensores alimentados são sensores que necessitam de alimentação elétrica externa para funcionar (24 VAC/DC, 230 VAC, tipicamente), ao contrário de sensores passivos que apenas variam uma resistência ou tensão. Um elemento é alimentado por uma tensão e a leitura do sensor é uma tensão pertencente a uma escala que, por vezes, pode ser alterada. A título de exemplo, um sensor de humidade pode indicar 0 VDC para uma humidade relativa de 0% e 10 VDC para 100% de humidade relativa [6].

Por fim, os sensores de comutação apenas indicam um estado (*ON/OFF*). São muito usados como dispositivos de segurança ou sinalização [6].

Outro equipamento da camada de campo que auxilia no controlo são os relés. Dispositivos que permitem através de um sinal elétrico, normalmente de baixa potência, acionar um interruptor eletromecânico para permitir a passagem de fontes de energia tipicamente com correntes mais elevadas. Alguns destes dispositivos têm fusíveis para proteção [6].

Atualmente, os sistemas são implementados com *Direct Digital Control*, em que os controladores executam a sua lógica (na camada de automação), de forma local, e enviam a informação necessária para um dispositivo central de gestão (na camada de

gestão). Contudo, ainda é possível encontrar instalações com sistemas *legacy*, com controlos analógicos ou pneumáticos que ainda existem, possivelmente, devido aos custos inerentes à atualização para um sistema atual [6].

2.2. Processo de comissionamento e normas aplicáveis

O comissionamento de edifícios é uma atividade que facilita o planeamento, conceção, construção, instalação, verificação e documentação, com o objetivo de assegurar a conformidade do Sistema de Gestão de Edifícios com os requisitos do projeto do edifício. É constituído por fases e atividades específicas, tanto para novas construções como para edifícios existentes. O processo inclui diversas tarefas, geralmente realizadas em equipa, para verificar e documentar a performance do edifício e verificar se a montagem está de acordo com os objetivos e critérios definidos. [8].

A atividade é frequentemente vista como a tarefa realizada após a construção do edifício e instalação de todos os equipamentos. Contudo, está presente ao longo de todo o processo de conceção, construção, entrega e otimização do desempenho das instalações [8].

O principal benefício do comissionamento de novas instalações e sistemas é a verificação, de forma documentada, do desempenho da instalação e dos sistemas, à medida que estes passam para a fase de operação e manutenção de longo prazo. O comissionamento, quando utilizado como um processo de identificação de problemas, pode também ser aplicado a instalações ou sistemas existentes, mesmo que não tenham sido comissionados inicialmente. Pretende-se garantir que o sucesso do projeto está claramente definido nos Requisitos do Projeto do Proprietário (*Owner's Project Requirements - OPR*) e que o edifício e os seus sistemas funcionam conforme o previsto [9].

O comissionamento serve de apoio ao projeto e contribui para uma instalação eficiente e segura, pois permite otimizar o consumo de energia e água, reduzir os custos operacionais, facilitar a orientação e formação da equipa de operação e manutenção e melhorar a documentação e o funcionamento dos sistemas instalados no edifício. Estas características podem resultar num aumento da rentabilidade, tanto ao nível da operação da instalação como do desempenho da sua equipa. O custo de não realizar o comissionamento equivale ao aumento de custos associados à correção posterior de falhas de projeto e construção, somado aos custos operacionais acrescidos. Por exemplo, em instalações consideradas críticas, o custo de não efetuar comissionamento pode ser medido pelo impacto do tempo de inatividade e da utilização inadequada das instalações [9].

Para além da eficiência energética e dos fatores relacionados com o desempenho global, outro fator que tem vindo a aumentar a procura pelo comissionamento é a intenção de obter certificação através de sistemas de avaliação de desempenho de edifícios. Estes sistemas de avaliação foram desenvolvidos com o objetivo de melhorar o planeamento, construção e desempenho em termos de eficiência energética, hídrica, condições ambientais, verificação, documentação e práticas operacionais [9].

2.2.1. Etapas do comissionamento

O processo de comissionamento deve ter início logo nas fases iniciais do projeto, prolongar-se até à entrada em funcionamento do edifício e dos sistemas técnicos e, até mesmo, durante toda a vida útil da instalação. Não se trata de uma função de conceção ou de construção, mas sim de um mecanismo de verificação e de garantia de qualidade, que assegura que os resultados de ambas as fases permitem alcançar os requisitos de desempenho estabelecidos pelo proprietário, abrangendo parâmetros como funcionalidade, eficiência energética, sustentabilidade, condições de conforto e resiliência [9].

É possível organizar o processo em cinco etapas comuns: planeamento, verificação de hardware, configuração, testes funcionais e validação final. Na figura 7 encontram-se as etapas do processo de comissionamento.

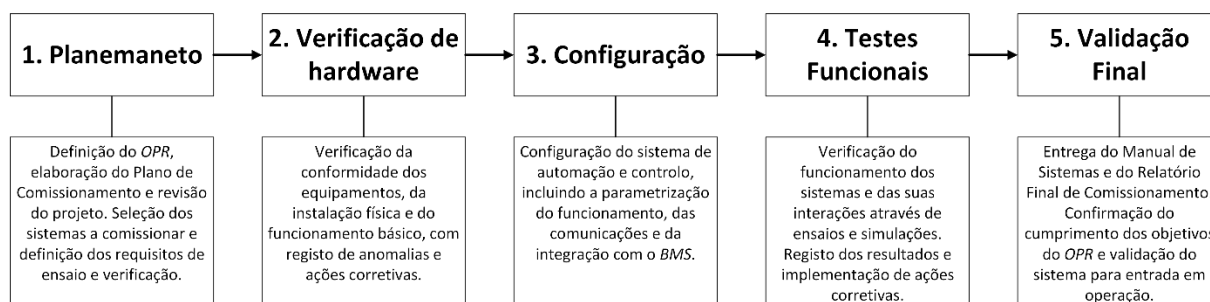


Figura 7. Fases do processo de comissionamento.

Na etapa de planeamento o comissionamento começa com o *OPR*, documento que define os objetivos de desempenho e serve como referência para todas as decisões subsequentes. Nesta mesma fase, inicia-se o Plano de Comissionamento, que descreve a estratégia global, bem como a estrutura inicial do Manual de Operação e Manutenção. Posteriormente, o plano é aprofundado e ajustado de acordo com os documentos do projeto. É aqui que são selecionados os sistemas a comissionar e são detalhados os requisitos de ensaio e de verificação. O *Commissioning Provider* realiza a revisão do projeto para assegurar a conformidade com o *OPR* e verifica a correta integração dos requisitos de comissionamento, emitindo recomendações fundamentadas e documentadas. Esta etapa de planeamento, constitui a base para o sucesso das fases seguintes [9].

A verificação de *hardware* centra-se na análise e aprovação de equipamentos e componentes, assegurando que cumprem os requisitos previamente definidos. Procede-se também às observações de campo para confirmar que a instalação física de sensores, controladores e restantes equipamentos está em conformidade com as especificações do projeto. Sempre que necessário, realizam-se testes para verificar o funcionamento básico de cada componente isoladamente. Durante esta fase é usado um registo de anomalias e de soluções, que documenta os problemas encontrados e as ações corretivas adotadas [9].

A fase de configuração corresponde à preparação dos sistemas de automação e controlo. Esta fase está situada entre a instalação física dos equipamentos e a realização dos testes funcionais. É nesta fase que se implementam as *sequences of operation* (descrições formais e detalhadas do modo como o sistema deve funcionar em diferentes condições), programação de alarmes, modos de funcionamento, e comunicações entre sistemas. A integração com a plataforma de gestão técnica do

edifício (*BMS*) é validada através da configuração de protocolos e interfaces, assegurando que o ambiente de teste reproduz as condições de operação previstas no plano de comissionamento [9].

A etapa de testes funcionais constitui o núcleo central do processo, sob a designação *Functional Performance Testing (FPT)*. Trata-se da verificação sistemática de que cada sistema, bem como as suas interações, satisfazem os critérios estabelecidos no *OPR*. Para tal, são realizados ensaios que abrangem modos normais e de falha, simulações de condições extremas e verificações da resposta dos sistemas de alarme e de redundância. Os resultados são registados em relatórios preliminares e, quando necessário, são realizadas ações corretivas [8] [9].

Finalmente, a etapa de validação final acontece após a conclusão substancial do projeto em que o proprietário assume formalmente a responsabilidade pelo edifício. Nesta fase realiza-se a entrega do Manual de Sistemas, que contém toda a documentação relevante, e elabora-se o Relatório Final de Comissionamento, contendo os registos das anomalias detetadas, as respetivas resoluções e eventuais situações em aberto aceites pelo proprietário. Quando aplicável, completam-se ainda os testes sazonais em condições reais de operação. Esta validação final confirma que os objetivos do *OPR* foram alcançados e constitui o encerramento formal do processo de comissionamento [9].

2.2.2. Normas

As normas para o comissionamento começaram em 1982, quando a *ASHRAE* formou um comité para documentar as boas práticas. Nessa altura foi lançado o *Guideline 1-1982 – The HVAC Commissioning Process*. Posteriormente, em 1996 foi feita uma atualização a este documento. Em 2007 surgiu a versão 1.1 que foi atualizada em 2025. Esta última edição trata da aplicação do processo de comissionamento aos novos sistemas de aquecimento, ventilação, ar condicionado e refrigeração (*HVAC&R*). O documento estabelece que o comissionamento deve iniciar-se desde o início do projeto e prolongar-se durante toda a vida útil da instalação. Em cada fase, define tarefas específicas destinadas a verificar que são cumpridos os Requisitos do Projeto do Proprietário. Fornece também acesso a anexos informativos online, com exemplos, formulários e materiais de referência aplicáveis a múltiplos contextos de comissionamento. Estes anexos representam boas práticas consensuais, mas não contêm requisitos obrigatórios do processo de comissionamento [10]. A *guideline* anterior serve de complemento para a *Guideline 0 - 2019 (The Commissioning Process)* e o *ANSI/ASHRAE/IES Standard 202 - 2024 (The Commissioning Process Requirements for New Buildings and New Systems)*. O *standard 202* conta com os requisitos mínimos para o comissionamento sem foco nos tipos específicos de edifícios, sistemas ou dimensões de projetos. O objetivo foi criar um processo mínimo aceitável de comissionamento para novos edifícios e sistemas. Já a *Guideline 0* apresenta as melhores práticas para a aplicação do comissionamento de edifícios inteiros (*whole-building commissioning*) em instalações, abrangendo tanto novos edifícios como projetos de renovação. Os princípios e processos descritos aplicam-se a todas as fases do projeto, proporcionando uma abordagem uniforme, integrada e consistente para o comissionamento. Inclui o *Total Building Commissioning Process (TBCxP)*, conforme definido pelo *National Institute of Building Sciences (NIBS)* na sua *Commissioning Process Guideline 0* [10] [11] .

Outra entidade que desenvolve uma série de guias e publicações técnicas que detalham as boas práticas e procedimentos para o comissionamento de sistemas prediais usadas internacionalmente é a *BSRIA (Building Services Research and Information Association)*. Entre os documentos recentes destacam-se [12]:

- *Commissioning Responsibilities Framework (BG 88/2025)*: fornece uma estrutura clara para definir papéis e responsabilidades nas tarefas do comissionamento;
- *Commissioning Air Systems (BG 49/2024)*: descreve procedimentos para testar e regular sistemas de ventilação, com tecnologias recentes e orientações atualizadas da indústria;
- *Commissioning Water Systems (BG 2/2010)*: fornece informações detalhadas para o comissionamento de sistemas de água em edifícios;
- *Model Commissioning Plan (BG 8/2009)*: apresenta um modelo de plano de comissionamento aplicável a projetos de construção, útil para estruturar tarefas, documentação e responsabilidades.

Os guias da *BSRIA* são reconhecidos e utilizados internacionalmente, especialmente, em conjunto com normas *CIBSE (Chartered Institution of Building Services Engineers)*. Servem como referência prática para complementar as normas globais como as da *ASHRAE*. Os documentos da *BSRIA* incluem ainda *checklists e templates* de documentos.

A *CIBSE* é uma organização profissional do Reino Unido dedicada à engenharia de instalações prediais. Desenvolve uma série de normas, códigos de prática e guias técnicos que abordam diversos aspetos do comissionamento de sistemas prediais, desde a conceção até a operação. Os *CIBSE Commissioning Codes* são documentos fundamentais que estabelecem procedimentos para o comissionamento de sistemas específicos, como distribuição de ar, sistemas hidráulicos e iluminação. Esses códigos fornecem orientações detalhadas sobre as etapas necessárias para garantir que os sistemas operam conforme o projetado, incluindo testes funcionais, regulação e documentação. Por exemplo, o *Commissioning Code A: Air Distribution Systems* aborda o comissionamento de sistemas de ventilação e de condutas e o *Commissioning Code M: Commissioning Management* oferece uma visão geral da gestão do processo de comissionamento em todas as fases do projeto [13].

Em Portugal, o Decreto-Lei n.º 101-D/2020 define os requisitos para a eficiência energética de edifícios novos e existentes, incluindo a instalação de sistemas de automatização e controlo (SACE), inspeções periódicas de sistemas técnicos (aquecimento, arrefecimento, ventilação), monitorização e registo de consumos de energia, integração com redes energéticas e mobilidade elétrica, e critérios de atribuição de incentivos financeiros baseados na melhoria do desempenho energético [14]. Contudo, não há nenhuma legislação que adapte as normas internacionais sobre o processo de comissionamento.

2.3. Principais subsistemas de um BMS

Existem vários sistemas que podem ser integrados num BMS. Serão abordados os mais comuns.

2.3.1. Sistema AVAC (Aquecimento, Ventilação e Ar Condicionado)

Um sistema AVAC pode ser dividido em duas categorias: sistemas *airside*, método principal de aquecimento e arrefecimento em edifícios, e sistemas *waterside*, que distribuem água quente e fria pelo edifício [6].

Nos sistemas *airside*, tem-se as *RTU* (*Rooftop Units*) e as *AHU* (*Air Handler Units*). Uma *RTU* é um sistema mais completo, geralmente instalado nos telhados dos edifícios, pois gera o ar condicionado (frio e quente) no mesmo equipamento. Nas *AHU* são necessárias fontes externas de energia térmica (tipicamente caldeiras e bombas de calor) provenientes, tipicamente, dos sistemas *waterside*. Em caso de falha nas fontes externas, várias *AHU* são afetadas, o que não acontece nas *RTU*, já que cada sistema tem a sua própria fonte. A razão para a utilização de *AHU* está na sua eficiência superior quando comparada com as *RTU*. As *AHU* ainda podem ser de volume de ar constante (*CAV*) ou de volume de ar variável (*VAV*) o que permite controlar a temperatura ou a pressão estática [6].

Existem alguns componentes base que compõem este tipo de sistema: *dampers*, válvulas, atuadores, ventoinhas e as serpentinas [6].

Os registos de ar estão presentes nas *AHU* de volume de ar variável. Permitem bloquear ou controlar o fluxo de ar fornecido a um determinado espaço ou entre os diferentes elementos da instalação [6].

As válvulas permitem fechar ou restringir a passagem de líquidos [6].

Os atuadores são dispositivos mecânicos que permitem movimentar os registos de ar e as válvulas. Tipicamente precisam de uma fonte de alimentação e um sinal de controlo [6].

As ventoinhas direcionam o fluxo de ar dentro do edifício. À medida que a velocidade da ventoinha varia, o fluxo de ar aumenta ou diminui. Contudo, este aumento ou diminuição não é linear. Existem diferentes gráficos que relacionam a velocidade da ventoinha com o volume de ar, dependendo do tipo de ventoinha [6].

As serpentinas permitem a transferência de calor de e para o ar. Geralmente, o conteúdo no interior das serpentinas é água quente ou fria, consoante o objetivo [6].

2.3.2. Sistema de iluminação

Ao longo dos anos, a iluminação em edifícios tem-se tornado cada vez mais complexa, sendo o principal objetivo a poupança energética e o conforto.

Um sistema de iluminação é composto, na sua base, por uma fonte, armadura de iluminação e interruptor. Numa camada superior existem os sistemas de controlo de iluminação [6].

Existem vários protocolos que permitem interligar o sistema de iluminação ao Sistema de Gestão de Edifícios. O mais comum é o *Dali*, que será explorado na secção dedicada aos protocolos, embora também possa ser usado o *BACnet/IP* [6].

Ao escolher um sistema de iluminação de um fabricante é necessário avaliar os protocolos que esse sistema suporta, quais os dados e controlos que o fabricante disponibiliza para integração no BMS e a priorização (definir a ordem de importância ou de ação de pontos e funções quando há conflitos, limites de recursos ou cenários de emergência) [6].

2.3.3. Sistema de gestão de energia

Outro fator importante num edifício é a medição do consumo energético e, mais importante, a gestão energética. Nos últimos anos, este sistema passou a incluir a gestão de fontes de energia renovável e armazenamento. Em caso de falha na rede externa, o sistema deve decidir quais as cargas mais importantes a manter em funcionamento com a energia disponível [6].

Para além disto, a monitorização em tempo real e análise de dados históricos permitem fazer a gestão de picos de consumo e avaliar a necessidade de melhorar a infraestrutura, como o banco de baterias ou painéis solares [6].

A gestão de energia relaciona a monitorização, o controlo e a conservação de energia. Existem três tarefas em que os sistemas de gestão de edifícios são usados para a gestão de energia. Estas tarefas são a medição, verificação e corte (ou redução) de carga [6].

A medição de energia é feita com recurso a medidores baseados em *IP* ou por outra ligação física ao sistema de gestão. Os dados recolhidos são armazenados numa base de dados [6].

O processo de verificação é um processo estruturado, tipicamente, pelo protocolo *IPMVP* (*International Performance Measurement and Verification Protocol*) para avaliar e comprovar poupanças energéticas obtidas após uma medida de eficiência. É necessário definir uma linha de base, medir consumos após a intervenção e calcular diferenças ajustadas a variáveis externas (clima, ocupação) [6].

No corte de carga, pretende-se diminuir o consumo energético, pela redução do consumo do edifício, pela forma como o sistema opera ou desligando alguns sistemas. Existem 3 níveis de corte de carga. No primeiro nível o gestor ajusta manualmente a performance de cada sistema conforme a necessidade imediata. No segundo nível o gestor define períodos ou condições específicas em que o sistema deve alterar o seu modo de funcionamento. No terceiro nível, o sistema desliga cargas (iluminação decorativa, climatização de áreas secundárias, entre outros) de forma autónoma, com base em parâmetros como potência contratada, picos de procura ou limites da rede. Esta tarefa surgiu para reduzir picos de consumo em momentos de elevada procura, mitigar as limitações da rede e evitar sobrecarga da infraestrutura elétrica, prevenindo apagões ou penalizações e incentivar práticas de eficiência energética. As cargas são programadas para funcionar em horários de menor custo ou com menor impacto na rede [6].

2.3.4. Sistemas de segurança

No contexto dos sistemas de gestão de edifícios, os sistemas de segurança são, geralmente, físicos. Os mais comuns são os sistemas de videovigilância compostos

pelas câmaras de videovigilância (a ligação pode ser por *IP* ou por cabo coaxial) e um servidor para armazenamento das imagens [6].

Para além do sistema referido no parágrafo anterior, podem ainda existir sistemas de alarme, controlo de acessos, e sistemas de deteção e prevenção de incêndios. Os sistemas de incêndio quando integrados num sistema *BMS* permitem iniciar medidas de segurança de forma rápida em conjunto com outros sistemas e não apenas gerar alertas. Por exemplo, o sistema de incêndio notifica o sistema de *BMS*, que comanda o sistema *AVAC* para executar a pressurização ou controlo de fumo de uma determinada área. Por sua vez, alguns dos equipamentos *AVAC* podem ter sensores de fogo/fumo conectados ao sistema *BMS* [6].

2.4. Protocolos usados em *BMS*

Os protocolos são *standards* usados para definir um método de transferência de informação por uma rede. Interessa primeiro definir o que é um protocolo aberto, pois existem vantagens, como o custo de utilização reduzido ou inexistente, que fazem com que a maioria dos protocolos usados sejam abertos. Existem essencialmente três critérios que são necessários cumprir [6]:

- O protocolo é um *standard* nacional ou internacional;
- O protocolo é interoperável, ou seja, o protocolo de comunicação é normalizado, o que permite que equipamentos de diferentes fabricantes troquem dados sem depender de *drivers* proprietários;
- Existirem, pelo menos, 5 fabricantes que usam o protocolo.

Existe uma grande quantidade de protocolos, contudo, apenas serão abordados os mais usados. Os protocolos abordados são todos considerados abertos, por respeitarem os critérios anteriores.

A figura 8 contém a integração dos protocolos dentro da arquitetura vista anteriormente. Será abordado um protocolo que permite a integração da camada de gestão com a camada de automação, o *BACnet*. Um protocolo que interliga a camada de automação com a camada de campo, o *KNX*, e três protocolos da camada de campo, *Modbus*, *Dali* e *M-bus*.

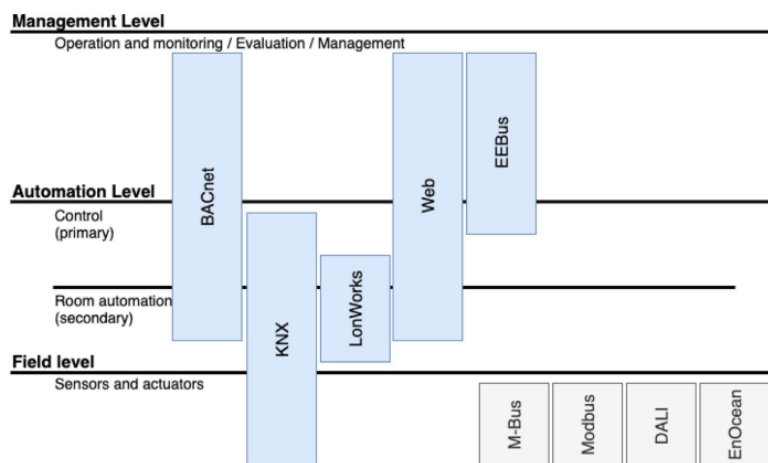


Figura 8. Integração dos protocolos na arquitetura de três camadas de um *BMS* [3].

2.4.1. Building Automation and Control Network (BACnet)

O *BACnet* é um protocolo desenvolvido pela *ASHRAE* (*American Society of Heating, Refrigerating and Air-Conditioning Engineers*) que fornece um *standard* para representar funções e operações de automação de edifícios e de controlo de dispositivos. Por exemplo, a partir do protocolo é possível fazer um pedido do valor de um sensor de humidade ou enviar um alerta do estado de uma bomba. Para cada dispositivo é criado um identificador e propriedades que refletem as funções e operações do equipamento [15].

Este protocolo pretende ser uma alternativa aos protocolos proprietários. Contém regras para a partilha de dados entre diferentes sistemas, define como é feita a partilha, quais os meios de comunicação/transmissão que podem ser usados e como é que a informação pode ser interpretada, com o objetivo de permitir a interoperabilidade dos sistemas e equipamentos de diferentes fabricantes [15] [16].

O *BACnet* foi desenvolvido a partir das convenções usadas na área do AVAC em que é usado o termo “pontos” para representar sensores, controlos de saídas, valores de controlo, entre outros, com diferentes características dependendo do fabricante. É através do protocolo que é definido um conjunto de objetos (os pontos) em que cada um dos objetos é composto por um conjunto de propriedades. É através destas propriedades que um objeto pode ser controlado por outros equipamentos *BACnet* [16].

A figura 9 contém um exemplo gráfico da representação de um objeto *BACnet*. Neste caso, o equipamento é um sensor, um termístor. As propriedades como a descrição, o tipo de equipamento e unidades são definidas no momento da instalação e configuração inicial. Outras propriedades não representadas na figura podem ser definidas pelo fabricante, como *Vendor_Name*, *Vendor_Identifier* e *Model_Name*. Neste exemplo, a leitura do *Present_Value* retornaria o valor 68,0 [16].

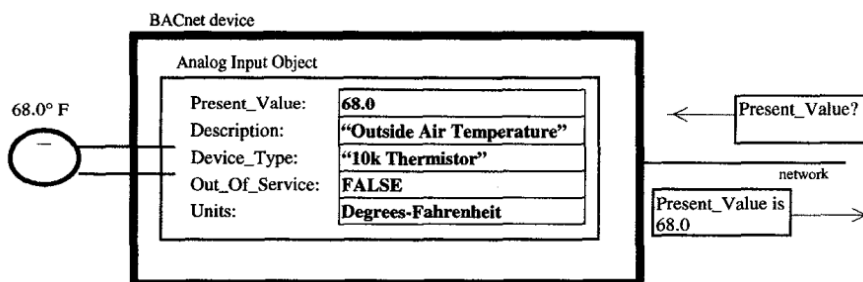


Figura 9. Objeto *Analog Input* [16].

A figura 10 contém os 18 objetos base *BACnet* definidos pela *ASHRAE*. A escolha dos objetos para um equipamento depende da sua função, sendo que nem todos os equipamentos devem ter todos os objetos. É comum que a maior parte dos equipamentos sejam compostos maioritariamente por objetos analógicos e binários. Contudo, cada equipamento deve ter o *Device Object* e as propriedades necessárias para descrevê-lo na rede. A propriedade *Object List* do *Device Object* contém uma lista de todas as propriedades de um determinado equipamento, ou seja, informação sobre o equipamento e as funcionalidades disponíveis e acessíveis a outros equipamentos na rede. Os fabricantes podem criar objetos proprietários que não são necessariamente entendidos ou aceitados por equipamentos de outros fabricantes [16].

Analog Input	Sensor input
Analog Output	Control output
Analog Value	Setpoint or other analog control system parameter.
Binary Input	Switch input
Binary Output	Relay output
Binary Value	Binary (digital) control system parameter
Calendar	Defines a list of dates, such as holidays or special events, for scheduling.
Command	Writes multiple values to multiple objects in multiple devices to accomplish a specific purpose, such as day-mode to night-mode, or to emergency mode.
Device	Properties tell what objects and services the device supports, and other device-specific information such as vendor, firmware revision, etc.
Event Enrollment	Describes an event which might be an error condition (e.g., "Input out of range") or an alarm that other devices might need to know about. It can directly tell one device, or use a Notification Class object to tell multiple devices.
File	Allows read and write access to data files supported by the device.
Group	Provides in a read single operation access to multiple properties of multiple objects.
Loop	Provides a standardized access to a "control loop."
Multi-state Input	Represents the status of a multiple-state process, such as a refrigerator's On, Off, Defrost cycle.
Multi-state Output	Represents the desired state of a multiple-state process (such as It's Time to Cool, It's Cold Enough. And It's Time to Defrost).
Notification Class	Contains a list of devices to be informed if an Event Enrollment object determines that a warning or alarm message needs to be sent.
Program	Allows a program running in the device to be started, stopped, loaded and unloaded, and reports the present status if the program.
Schedule	Defines a weekly schedule of operations (performed by writing to specified list of objects) with exceptions such as holidays. Can use a Calendar object for the exceptions.

Figura 10. Objetos *BACnet* padrão [16].

Existem 123 propriedades padrão. Num objeto algumas das propriedades são opcionais, enquanto outras são obrigatórias. As propriedades têm comportamentos característicos definidos pelas suas especificações, em especial as que estão associadas a alarmes e notificações de eventos. Não é permitido adicionar propriedades aos objetos, exceto nos objetos proprietários [16].

A figura 11 contém a lista de propriedades de um objeto *Analog Input*, que poderia ser do equipamento da figura 9. Algumas propriedades como *Low_Limit* e *High_Limit* podem emitir notificações e alarmes quando os limites são ultrapassados. Já as propriedades *Object_Name*, *Object_Identifier* e *Object_Type* devem estar presentes em qualquer objeto *BACnet* [16].

Object_Identifier	Required	Analog Input #1
Object_Name	Required	"AI 01"
Object_Type	Required	Analog Input
Present_Value	Required	68.0
Description	Optional	"Outside Air Temperature"
Device_Type	Optional	"10k Thermistor"
Status_Flags	Required	In_Alarm, Fault, Overridden, Out_Of_Service flags
Event_State	Required	Normal (plus various problem-reporting states)
Reliability	Optional	No_Fault_Detected (plus various fault conditions)
Out_Of_Service	Required	False
Update_Interval	Optional	1.00 (seconds)
Units	Required	Degrees-Fahrenheit
Min_Pres_Value	Optional	-100.0 , minimum reliably read value
Max_Pres_Value	Optional	+300.0 , maximum reliably read value
Resolution	Optional	0.1
COV_Increment	Optional	Notify if Present_Value changes by increment: 0.5
Time_Delay	Optional	Seconds to wait before detecting out-of-range: 5
Notification_Class	Optional	Send COV notification to Notification Class Object: 2
High_Limit	Optional	+215.0 , Upper normal range
Low_Limit	Optional	-45.0 , Lower normal range
Deadband	Optional	0.1
Limit_Enable	Optional	Enable High-limit-reporting , Low-limit-reporting .
Event_Enable	Optional	Enable To_Offnormal , To_Fault , To_Normal change reporting.
Acked_Transitions	Optional	Flags indicating received acknowledgments for above changes.
Notify_Type	Optional	Events or Alarms

Figura 11. Propriedades de um objeto *Analog Input* [16].

Os serviços são métodos que permitem a um equipamento *BACnet* adquirir informação de outros equipamentos, fazer com que um equipamento execute alguma ação ou notifique um ou mais equipamentos sobre algum evento. Cada *Service Request* enviado tem um *Service Reply* como resposta com dados que devem ser sobre a informação solicitada. Só existe um tipo de serviço obrigatório que um equipamento deve ter, que é o *ReadProperty*. Este serviço permite a leitura de propriedades de outros equipamentos [16].

A comunicação numa rede *BACnet* pode ser do tipo *request-response* ou através da gestão de eventos. Ambas baseadas no princípio cliente-servidor. Existem 5 grupos de serviços que constituem 35 tipos de mensagens [15] [17]:

- Acesso e manipulação das propriedades dos objetos (*Object Access Services*): mensagens relacionadas com a leitura, escrita e alteração de propriedades definidas nos objetos *BACnet*;
- Alarmes e eventos (*Alarm and Event Services*): usam mecanismos para alertar subscritores desses alarmes e eventos sobre alterações no sistema através de notificações, confirmação e gestão de condições anómalas;
- Carregamento e transferência de ficheiros (*File Access Services*): serviços de leitura e escrita de ficheiros entre equipamentos de forma remota;
- Gestão do funcionamento de equipamentos remotos (*Remote Device Management Services*): comandos de arranque, reinício, sincronização de tempo, entre outros. Contém as mensagens *WhoIs* e *IAm* para identificar equipamentos e objetos na rede. Esta identificação é feita através de mensagens de *Broadcast*;
- Funções de terminal virtual (*Virtual Terminal Services*): simulação de um terminal de texto para comunicação interativa.

Existe ainda um mecanismo de resolução de conflitos e prioridades. O protocolo é baseado em pares (*peer-based*) orientado a objetos permitindo que qualquer equipamento na rede interaja com propriedades de outros equipamentos. Como podem existir vários dispositivos a pretender alterar um parâmetro de um equipamento, existe um método para resolver conflitos baseado num esquema de prioridades [18].

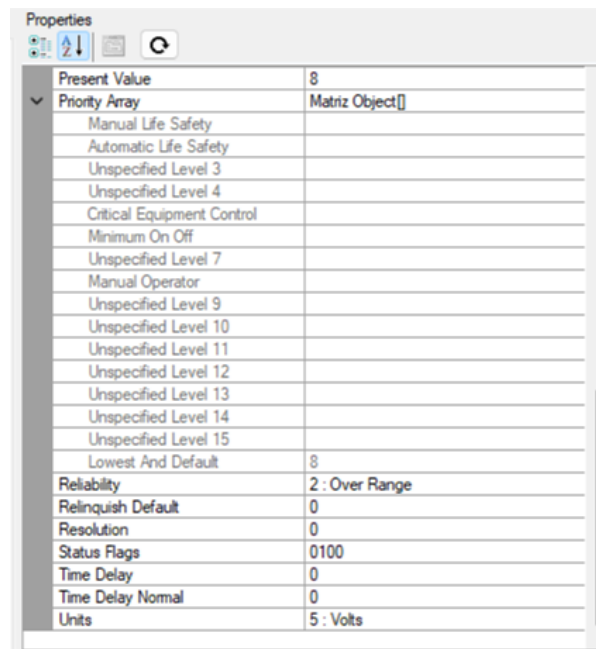
Nem todas as propriedades têm o sistema de prioridades. Existem dois tipos de propriedades: *writable properties* e *commandable properties*. As primeiras são propriedades que não possuem um mecanismo de prioridade associado, desta forma, qualquer nova escrita substitui o valor anterior. Já as *commandable properties* são aquelas cujas alterações de valor desencadeiam ações físicas, como o *Present_Value* das saídas analógicas ou binárias. Estas propriedades utilizam obrigatoriamente um *array* de prioridades [18].

O *array* de prioridades consiste numa estrutura de dados com 16 níveis, onde o valor da prioridade é inversamente proporcional à sua importância: o nível 1 representa a prioridade mais alta (geralmente reservada para segurança crítica), enquanto o nível 16 representa a prioridade mais baixa. O sistema de prioridades segue os seguintes princípios [18]:

1. Escrita de valores: Quando um equipamento envia um pedido *WriteProperty*, deve especificar o identificador da propriedade, o valor pretendido e o nível de prioridade associado;

2. Determinação do valor ativo: O equipamento que recebe o comando avalia o *array* e executa o *PresentValue* no nível preenchido de maior prioridade (o número mais baixo que não seja *NULL*);
3. Mecanismo de *Relinquish*: Quando um cliente deixa de controlar uma prioridade, envia um comando de *Relinquish*, em que define o valor naquele nível de prioridade específico como *NULL*;
4. Valor por defeito: Caso todos os 16 níveis do *array* de prioridades estejam preenchidos com *NULL*, a propriedade assume automaticamente o valor configurado na propriedade *RelinquishDefault* do objeto.

A figura 12 contém algumas das propriedades de um objeto *BACnet* visíveis através do programa *YABE*. Nela é possível visualizar as propriedades referidas anteriormente. Neste caso, há um valor escrito na prioridade mais baixa. Sendo assim, o *Present Value* adquire esse valor. Se nenhuma prioridade indicasse nenhum valor, o *Present Value* iria adquirir o valor da propriedade *Relinquish Default*, 0, neste caso [18].



Property	Value
Present Value	8
Priority Array	Matriz Object[]
Manual Life Safety	
Automatic Life Safety	
Unspecified Level 3	
Unspecified Level 4	
Critical Equipment Control	
Minimum On Off	
Unspecified Level 7	
Manual Operator	
Unspecified Level 9	
Unspecified Level 10	
Unspecified Level 11	
Unspecified Level 12	
Unspecified Level 13	
Unspecified Level 14	
Unspecified Level 15	8
Lowest And Default	8
Reliability	2 : Over Range
Relinquish Default	0
Resolution	0
Status Flags	0100
Time Delay	0
Time Delay Normal	0
Units	5 : Volts

Figura 12. Propriedades de um objeto *BACnet* vistas no *YABE*.

Apesar de um equipamento conseguir escrever em qualquer prioridade, desde que já não esteja ocupada, algumas prioridades devem ser usadas para aplicações específicas que constam na tabela 1. As restantes prioridades podem ser atribuídas livremente de acordo com as necessidades específicas de cada instalação ou fabricante. A prioridade 1 é a mais elevada do sistema, destina-se a comandos de emergência iniciados manualmente por um operador humano em situações onde a integridade física de pessoas ou do edifício está em risco imediato. A prioridade 2 é reservada para sistemas automáticos de emergência (como sistemas de deteção e alarme de incêndio). A prioridade 5 é usada por equipamentos que não podem sofrer alterações por processos de rotina. A prioridade 6 é frequentemente usada para implementar salvaguardas de proteção de equipamentos, como tempos mínimos de funcionamento ou de paragem de compressores, evitando o desgaste prematuro ou danos resultantes de ciclos de ligar/desligar demasiado frequentes. A prioridade 8 é destinada a alterações realizadas por um utilizador humano tipicamente através de

uma interface de gestão (*HMI*) em condições normais de funcionamento que não sejam de emergência, normalmente operações de manutenção [18].

Tabela 1. Aplicações de cada prioridade do protocolo *BACnet* [18].

Prioridade	Aplicação Padrão
1	Segurança (Manual)
2	Segurança (Automático)
3	Disponível
4	Disponível
5	Controlo de Equipamento Crítico
6	Mínimo <i>on/off</i>
7	Disponível
8	Operação Manual
9 a 16	Disponível

A norma atual não define como é que o sistema deve agir quando vários equipamentos tentam escrever no mesmo nível de prioridade com valores diferentes, o que pode levar a cenários em que escreve o primeiro que conseguir, o que vai contra o propósito inicial de organização da escrita em prioridades [18].

O *BACnet* pode operar sobre diferentes tipos de rede, incluindo *Ethernet*, *ARCNET*, *MS/TP* (*master-slave/token-passing*) e *PTP* (*point-to-point*, usado em linhas telefónicas ou ligações série *EIA-232*). Contudo, o *ARCNET* é considerado obsoleto e o *PTP* tem utilização muito limitada [15].

Reconhecendo a importância do protocolo *IP* nas comunicações modernas, a *ASHRAE* desenvolveu inicialmente o método *Annex H.3*, que permitia enviar num túnel o tráfego *BACnet* através de redes *IP*, encapsulando os dados *BACnet* em pacotes *IP*. Este método exigia, porém, a presença de *gateways* em cada sub-rede, responsáveis por empacotar e desempacotar a informação, o que introduzia complexidade e limitações. Para superar essa limitação, surgiu o *Annex J – BACnet/IP*, que permite a transmissão de mensagens *BACnet* diretamente em formato nativo *IP*. Esta evolução trouxe maior flexibilidade e escalabilidade, permitindo a integração mais simples de sistemas *BACnet* em infraestruturas de rede baseadas em *IP* [15].

O *BACnet/IP* costumava ser usado apenas na camada de gestão. Contudo, houve um aumento dos controladores que passaram a suportar nativamente comunicações por *IP*, o que permitiu a utilização do protocolo na camada de automação [6].

Para localizar os dispositivos na rede, o *BACnet/IP* usa mensagens de *broadcast* que, por norma, não atravessam *routers*. Para contornar a situação é necessário usar um dispositivo chamado *BACnet Broadcast Management Device (BBMD)* que tem como único objetivo manter uma lista de outras redes *BACnet*, caso seja necessário mandar um *broadcast* para um dispositivo noutra rede [6].

O *BACnet MS/TP* é usado maioritariamente para comunicações entre controladores de campo. As redes *MS/TP* usam uma topologia em barramento em que um controlador é conectado ao outro por 3 a 4 fios. Os controladores comunicam ao

partilhar uma mensagem especial chamada *token*. Apenas o controlador com o *token* pode enviar mensagens [6].

Antigamente, apenas a segurança física era suficiente, contudo, atualmente os edifícios estão sujeitos às ameaças e vulnerabilidades das redes *IP* existentes. A principal resistência à adoção de mecanismos de segurança era o *overhead* computacional, o que nos dias de hoje já não é um argumento válido para não implementar medidas adequadas [17].

O *BACnet* tradicional não garante, por si só, uma proteção adequada face aos padrões de segurança atuais. Logo, o *IT-WG* (*Eurosai IT Working Group*) desenvolveu uma nova proposta centrada em mecanismos de comunicação segura que recorrem exclusivamente a boas práticas bem estabelecidas. A nova tecnologia recebeu a designação de *BACnet Secure Connect* (*BACnet/SC*), que fornece os meios necessários para estabelecer ligações de comunicação seguras entre dispositivos de automação em edifícios, tanto em ambientes *cloud*, como dentro de instalações físicas. Este protocolo utiliza as técnicas mais recentes de segurança ao mesmo tempo que mantém compatibilidade com todas as implementações e dispositivos existentes [19].

O *BACnet/SC* não depende de mensagens de *broadcast*, característica que o torna mais eficiente e alinhado com as práticas de rede atuais. Adicionalmente, elimina a necessidade do *BACnet/IP Broadcast Management Device* (*BBMD*), bem como da respetiva configuração. Por fim, destaca-se a sua compatibilidade nativa com os dispositivos de *firewall* comuns nas infraestruturas de rede atuais, o que facilita a integração e garante maior robustez no cumprimento dos requisitos de segurança e gestão de tráfego da rede [19].

O protocolo permite o transporte seguro de mensagens através do *IP Secure WebSocket*, que constitui uma extensão do *HTTPS* sobre o *Transport Layer Security* (*TLS*). Esta solução assegura elevados níveis de proteção na comunicação [19].

2.4.2. Modbus

O *Modbus* é um protocolo de comunicação publicado pela *Modicon* em 1979. À data, o protocolo tinha como principal objetivo a comunicação com controladores lógicos programáveis (*PLC*) fabricados pela própria *Modicon* e aplicados em sistemas de automação industrial. Atualmente, a *Modicon* pertence à *Schneider Electric* e, em 2004, a gestão da norma *Modbus* foi transferida para uma organização sem fins lucrativos, a *Modbus-IDA*, composta maioritariamente por utilizadores e fornecedores da indústria de automação [15].

O *Modbus* constitui um protocolo de mensagens na camada de aplicação do modelo *OSI*, concebido para a comunicação no modelo cliente-servidor entre dispositivos ligados por diferentes tipos de barramentos ou redes. Pode ser implementado em redes *Ethernet*, em transmissões série assíncronas como o *RS-232*, *RS-485* ou ainda em redes de alta velocidade com passagem de testemunho, designadas *Modbus Plus*. Nas implementações sobre *Ethernet* ou *Modbus Plus*, as mensagens *Modbus* são encapsuladas (*tunneled*) na estrutura de tramas ou pacotes do respetivo meio de transmissão [15].

A implementação mais comum utiliza a camada física *RS-485*, através do *Modbus RTU* (representação binária dos dados) ou a *Modbus ASCII* (representação legível). Na variante *Ethernet*, adota-se o *Modbus/TCP* [15].

Tal como outros protocolos de comunicação, o *Modbus* define uma estrutura de mensagens e o formato dos respetivos campos, incluindo a forma como um controlador solicita acesso a outro dispositivo, os mecanismos de resposta, os métodos de deteção e aviso de erros, a identificação inequívoca dos dispositivos e o reconhecimento de mensagens destinadas a cada unidade [15].

As diferentes versões apresentam funcionalidades distintas. O *Modbus* básico segue um modelo mestre-escravo, em que os dispositivos escravos não comunicam autonomamente com o mestre, a menos que este os interroge. Já na implementação em *Ethernet*, os dispositivos *Modbus/TCP* podem enviar mensagens ao mestre sem necessidade de inquérito prévio [15].

As implementações tradicionais estão limitadas a 247 dispositivos numa rede, embora essa restrição não se aplique ao *Modbus/TCP* [15].

O *Modbus RTU* é a forma mais comum de comunicação, realizada através de uma porta série, em que os dispositivos estão organizados numa topologia linear. Cada dispositivo tem um identificador único e um conjunto de registos num ficheiro de registos ou pontos. O ficheiro de registos é adicionado ao *supervisory device* [15].

O *Modbus TCP* permite transmitir dados pela rede. Geralmente é usado em implementações de larga escala. O protocolo permite estabelecer de forma fiável comunicação com dispositivos noutras redes. Destaca-se por ser um protocolo simples, mas eficaz, amplamente utilizado em automação e energia. Contudo, os projetistas enfrentam desafios recorrentes, como latências elevadas e problemas de temporização na comunicação [6] [15].

2.4.3. KNX

O *KNX*, também conhecido por *European Installation Bus (EIB)* é um sistema de comunicação e controlo de edifícios. Conecta, essencialmente, dispositivos como sensores, atuadores, controladores e monitores. Foi desenvolvido para ser usado na instalação elétrica para implementar funções de automação em edifícios [20].

A figura 13 mostra a estrutura base de comunicação com o protocolo para, por exemplo, um sensor e um atuador que é feita por um barramento, sem a necessidade de um dispositivo central. O sensor cria um sinal e envia esse sinal para o atuador. O atuador, por sua vez, envia um sinal de *acknowledge* e executa uma ação de acordo com o sinal recebido [20].

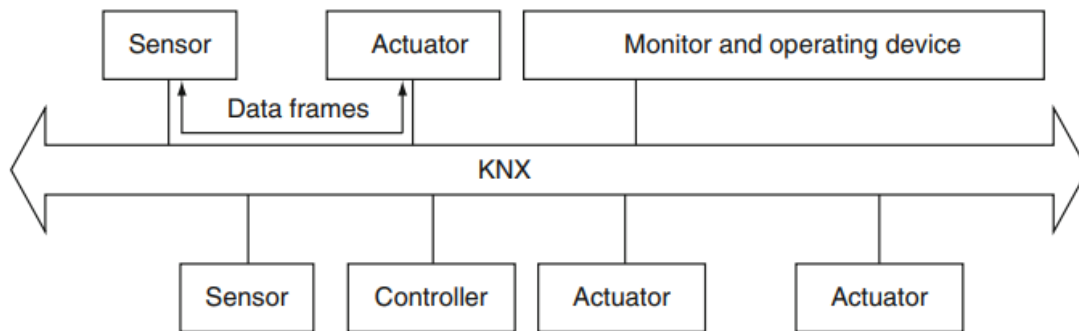


Figura 13. Estrutura base de funcionamento do protocolo *KNX* [20].

O protocolo permite que dispositivos de diferentes fabricantes possam comunicar entre si. Infelizmente, dispositivos que utilizam este protocolo costumam ter um preço superior aos dispositivos de instalações convencionais [20].

Através de *gateways*, sistemas *KNX* podem ser integrados na infraestrutura de rede de um edifício (redes *TCP/IP*), permitindo que sejam acedidos de forma externa pela Internet [20].

Diferentes tipos de meios podem ser usados para transferir dados de um dispositivo para outro: par entrançado de cobre (*KNX.TP*), *powerline* (*KNX.PL*), rádio frequência (*KNX.RF*), *Ethernet* (*KNXnet/IP*). Alguns fabricantes já começaram a implementar transmissão por fibra ótica. O meio mais usado continua a ser o *KNX.TP*, por ser a alternativa mais barata. Neste caso, são usados dois fios que servem para alimentação e transmissão de informação, sendo que pode haver um par de fios extra para reserva ou redundância [20]. A Figura 14 contém um exemplo de um equipamento com suporte ao protocolo.



Figura 14. Exemplo de um interruptor com suporte a *KNX* [20].

Tipicamente existe uma fonte de alimentação que fornece 30 *VDC* ao barramento. Dependendo da dimensão do sistema, podem ser necessários acopladores e repetidores de linha [20].

Os dispositivos devem ter um endereço físico único e podem ainda ter um endereço de grupo para permitir alterar o estado de um conjunto de equipamentos [20].

Como a comunicação é descentralizada, não depende de um controlador central, cada dispositivo (sensores, atuadores, controladores) tem inteligência própria e comunica diretamente com os outros [20].

Para garantir a interoperabilidade, os equipamentos de diferentes fabricantes devem ser certificados pela *KNX Association* [20].

Para que possa haver comunicação com a camada de gestão é necessário haver um gateway, por exemplo, de *KNX* para *BACnet*. O protocolo é, essencialmente, para comunicação entre equipamentos da camada de campo e a camada de automação [20].

2.4.4. Meter-bus (M-bus)

A integração transparente e de fácil implementação é um requisito essencial nos sistemas de *smart metering*. As soluções *wireless* são ideais devido à sua flexibilidade, embora soluções cabladas também possam ser consideradas quando a infraestrutura já se encontra disponível. O *M-Bus*, normalizado na *EN 13757*, especifica a comunicação para contadores de consumo (água, gás e eletricidade), bem como para concentradores de dados (*data concentrators*) destinados à recolha da informação de medição. Trata-se do padrão mais usado no setor da medição, não apenas na Europa, mas também em muitas outras regiões [21].

O protocolo apresenta as seguintes vantagens [21]:

- Cumpre a norma *EN 13757* e é compatível com a *Open Metering System (OMS)*;
- Baixo consumo energético e baixa complexidade, permitindo grande autonomia dos dispositivos;
- Protocolo robusto que opera em banda estreita, com potência de transmissão até 500 mW (*Wireless M-Bus*), sendo fiável também na versão com fios;
- Elevada escalabilidade com fácil expansão de rede;
- Protocolo considerado seguro com suporte para a encriptação *AES-128*, garantindo confidencialidade e integridade dos dados transmitidos;
- Bom equilíbrio entre desempenho e preço.

A tecnologia *Wireless M-Bus* assenta, na maioria dos casos, numa topologia em estrela, o que possibilita uma estrutura de rede simples e eficiente. Trata-se de uma solução flexível, facilmente expansível e de atualização pouco complexa, permitindo a substituição de dispositivos convencionais por contadores inteligentes sem necessidade de alterações significativas na infraestrutura existente. A comunicação é realizada através de transmissões periódicas e automáticas, o que dispensa a solicitação por parte de um elemento central [21].

Um dos principais atributos desta tecnologia é o seu baixo consumo energético, característica que a torna particularmente adequada para equipamentos alimentados a bateria, permitindo o aumento da sua vida útil. Embora a cobertura da rede possa ser limitada em determinados contextos, essa restrição pode ser superada mediante a integração com tecnologias complementares, como o *LoRaWAN* ou o *Mioty*, aumentando assim o alcance e a robustez da comunicação. [21].

O *Wired M-Bus* baseia-se no modelo clássico mestre-escravo, em que os contadores apenas transmitem dados quando solicitados pelo mestre. Neste tipo de configuração, os dispositivos permanecem permanentemente em modo de recepção ativa (*RX*), garantindo disponibilidade imediata para responder a pedidos. Esta característica torna o *Wired M-Bus* mais adequado para sistemas ligados à rede elétrica, uma vez que a operação com recurso a uma bateria seria demasiado exigente em termos de consumo energético [21].

No contexto do *Smart Metering*, o *Wired M-Bus* apresenta vantagens significativas, sobretudo pela sua elevada imunidade a interferências, resultante da transmissão de dados por cabo. Contudo, este modelo envolve limitações de ordem logística, dado que em muitos cenários de utilização é necessário interligar um grande número de dispositivos distribuídos por longas distâncias, o que implica a instalação de extensas redes de cabos. Por este motivo, a transmissão de dados via *Wired M-Bus* é recomendada principalmente em situações em que a infraestrutura de cabos já se encontra disponível ou quando a sua implementação seja economicamente viável. Uma vez instalada, a rede caracteriza-se por uma manutenção mínima, elevada rapidez na aquisição de dados e eficiência na leitura remota de consumos [21].

Relativamente à topologia, o *Wired M-Bus* recorre preferencialmente à topologia em barramento (*bus*), por razões de fiabilidade e custo-eficiência. A rede é constituída por um mestre, responsável pela lógica central de comunicação, um conjunto de escravos (os contadores de consumo) e um cabo de dois condutores que constituem o meio físico de transmissão. Os escravos são conectados paralelamente ao cabo, partilhando o mesmo meio de comunicação, sendo o mestre responsável pelo controlo integral das trocas de dados, interrogando sequencialmente cada escravo no sistema hierárquico [21].

2.4.5. Digital Addressable Lighting Interface (DALI)

O *DALI* (*Digital Addressable Lighting Interface*) é atualmente reconhecido como o padrão global para controlo de iluminação, sendo suportado por uma vasta gama de produtos disponibilizados por todos os principais fabricantes. Trata-se de um protocolo aberto, promovendo assim a interoperabilidade no setor. A certificação obrigatória no âmbito do *DALI-2* assegura compatibilidade entre equipamentos de diferentes fabricantes [22].

Entre as principais características do *DALI* destaca-se a instalação simplificada, uma vez que os cabos de alimentação e de controlo podem ser instalados em conjunto, sem a necessidade de blindagem. A comunicação é digital, o que permite que múltiplos dispositivos recebam exatamente os mesmos valores de regulação luminosa (*dimming*), o que assegura um desempenho estável e preciso. Cada dispositivo possui ainda um endereço único, característica que possibilita um controlo altamente flexível e sofisticado. O *DALI* é ainda endereçável, permitindo a criação de grupos, definição de cenários e adaptação dinâmica de sensores e interruptores ao espaço. A sua curva de regulação é normalizada, assegurando plena interoperabilidade entre fabricantes. Para além destas vantagens, o *DALI* suporta funcionalidades adicionais, como controlo de cor, testes de emergência e gestão de cenários complexos [22].

A primeira versão do protocolo apresentava, contudo, algumas limitações, entre as quais se destacam a reduzida dimensão da trama de dados (16 bits: 8 bits para endereçamento e 8 bits para comando) e a ausência de mecanismos de deteção de

colisões. Estas restrições levaram ao desenvolvimento de adaptações proprietárias que acabaram por comprometer a interoperabilidade. Com o *DALI-2*, normalizado pela *IEC 62386*, estas limitações foram ultrapassadas, sendo agora garantida a conformidade através da certificação supervisionada pela *Digital Illumination Interface Alliance (DiiA)*. Produtos que têm o logótipo *DALI-2* cumprem integralmente a norma e asseguram compatibilidade. É possível integrar dispositivos *DALI* e *DALI-2* numa mesma instalação, embora com algumas restrições, sendo os *drivers LED* o exemplo mais comum desta interoperabilidade [22].

Ao comparar este protocolo com protocolos de gestão predial como o *KNX* e *BACnet*, as diferenças tornam-se evidentes. Estes últimos foram concebidos para integrar múltiplos sistemas (*HVAC*, segurança, controlo de acessos, elevadores, entre outros), não sendo, por norma, adequados para controlo de iluminação, pois existem poucos *drivers LED* com ligação direta. Em contrapartida, o *DALI/DALI-2* foi concebido exclusivamente para iluminação, com comandos específicos. Em muitas implementações, o *BMS* recorre a *KNX* ou *BACnet* para controlo global do edifício, enquanto o *DALI* é reservado para a gestão da iluminação. A integração entre sistemas é assegurada através de *gateways*, permitindo, por exemplo, que a iluminação de corredores controlada via *DALI* seja acionada em resposta a um alerta proveniente do sistema de segurança [22].

Em termos físicos, o *DALI* é constituído por um par de condutores que transporta sinais digitais de controlo desde os dispositivos de entrada (como sensores e interruptores) até ao controlador de aplicação. Este, por sua vez, aplica as regras previamente programadas para emitir sinais de saída destinados aos dispositivos terminais, como *drivers LED*. Trata-se, portanto, de um sistema de comunicação estruturado que garante flexibilidade e precisão no controlo de iluminação [22]. A figura 15 contém um esquema base da implementação de um sistema com recurso a equipamentos *DALI*.

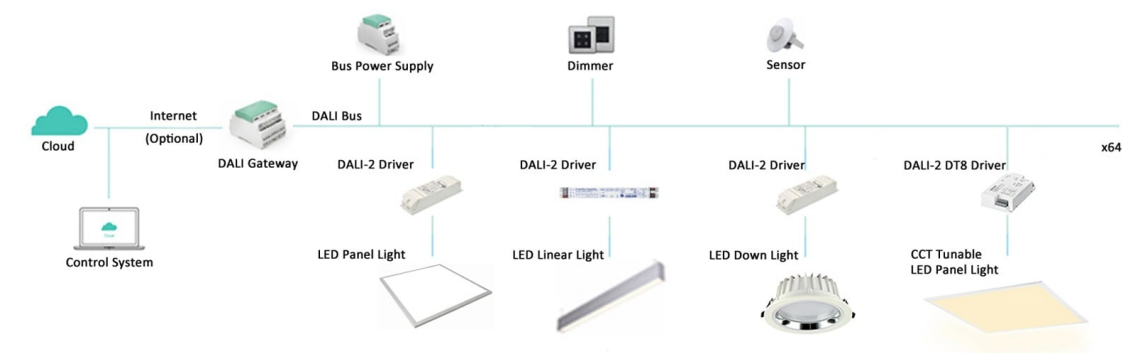


Figura 15. Arquitetura base de um sistema de iluminação *DALI* [23].

A unidade de alimentação do barramento (*PSU*) fornece e assegura a estabilidade da tensão no barramento [22].

Os equipamentos de iluminação, como *drivers LED*, são elementos indispensáveis em qualquer sistema *DALI*. Estes recebem comandos diretamente do barramento e respondem de acordo com os comandos recebidos. Os *drivers* podem ser compatíveis com *DALI* e/ou *DALI-2* [22].

Os dispositivos de entrada, como sensores e interruptores, comunicam com o controlador de aplicação através de tramas de dados de 24 bits. Importa sublinhar que

estes não interagem diretamente com os dispositivos de controlo de iluminação, mas apenas com o controlador [22].

O conceito de instâncias é também relevante. Um único dispositivo, como um sensor, pode integrar múltiplas funções (por exemplo, um detetor de movimento *PIR*, um sensor de luminosidade e um recetor de infravermelhos). Cada uma destas funções é designada instância. No *DALI-2*, cada instância pode ser atribuída a diferentes grupos de controlo, permitindo um endereçamento independente e uma gestão mais flexível da iluminação [22].

Por fim, os dispositivos de controlo, designados controladores de aplicação, representam o “cérebro” do sistema. Estes recebem mensagens de 24 bits provenientes dos dispositivos de entrada e convertem-nas em comandos de 16 bits dirigidos aos *drivers*. Além disso, são responsáveis pela gestão do tráfego de dados no barramento, detetando colisões e procedendo à retransmissão de comandos sempre que necessário [22].

O *DALI* suporta várias configurações de rede, podendo ser implementado em topologia de barramento, estrela, árvore ou linha, ou em combinações destas. Não é, contudo, compatível com topologias em anel ou em malha (*mesh*) [22].

O barramento funciona com uma tensão de aproximadamente 16 V, não sendo classificado como *SELV* (*Safety Extra Low Voltage*). Por este motivo, a instalação deve recorrer sempre a cabos com especificação adequada para a rede de alimentação. Outra característica relevante é que o barramento não é sensível à polaridade. Os dois condutores podem ser ligados indiferentemente, sendo habitualmente identificados nos terminais dos dispositivos como *DA*, *DA* (*Digital Addressable*, dois terminais dedicados ao barramento de comunicação) [22].

Adicionalmente, não é necessária blindagem dos cabos. A baixa taxa de transmissão de dados (cerca de 1200 símbolos por segundo), a tensão relativamente elevada do barramento e a tolerância alargada a flutuações de tensão conferem ao sistema uma robustez significativa contra interferências elétricas.

Na prática, os cabos utilizados são frequentemente multicondutores, capazes de transportar simultaneamente a alimentação principal. Os exemplos mais comuns incluem:

- cabos de 5 condutores: fase, terra, neutro, *DA*, *DA*;
- cabos de 6 condutores: fase, fase de emergência (iluminação de emergência), terra, neutro, *DA*, *DA*.

2.5. Ferramentas digitais de apoio ao comissionamento

As ferramentas de comissionamento podem ser divididas em 2 categorias: ferramentas gerais e ferramentas de teste e monitorização.

As ferramentas gerais permitem, essencialmente, fazer a gestão do projeto, gerir documentos e o planeamento. Ou seja, estas ferramentas destinam-se a abordar o comissionamento durante a vida útil do projeto. A título de exemplo, é de destacar ferramentas como o *Autodesk Construction Cloud (BIM 360)* e *Facility Grid*.

As ferramentas de teste e monitorização são usadas, normalmente, na fase de testes funcionais e validação final a fim de avaliar o bom funcionamento da instalação. Podem ser usadas apenas num contexto específico, ou de aplicação geral. Dada a grande variedade de ferramentas disponíveis no mercado, apenas serão analisadas três ferramentas com funções distintas que representam a maior parte do *software* disponível atualmente.

Apesar da existência de outras ferramentas comerciais e académicas destinadas ao apoio ao comissionamento, a seleção apresentada neste trabalho incidiu sobre o *WebCTRL*, o *EcoStruxure Building Commission* e o *AutoCXI* por constituírem soluções representativas de diferentes abordagens atualmente utilizadas no setor. O *WebCTRL* representa uma plataforma *BMS* com funcionalidades integradas de apoio ao comissionamento; o *EcoStruxure Building Commission* corresponde a uma ferramenta desenvolvida por um fabricante de referência na área da automação de edifícios diretamente relacionada com o ambiente tecnológico utilizado durante o estágio e o *AutoCXI* representa uma solução especializada em comissionamento automatizado, independente de um fabricante específico. Em conjunto, estas ferramentas permitem comparar diferentes estratégias de implementação, níveis de automatização e limitações das soluções atualmente disponíveis, servindo de base para identificar oportunidades de inovação exploradas no presente trabalho.

2.5.1. *WebCTRL*

O sistema de automação de edifícios *WebCTRL* é uma plataforma robusta, baseada na web, que oferece aos gestores das instalações um conjunto de ferramentas que permitem implementar medidas de gestão de energia, identificar problemas operacionais críticos e analisar os respetivos resultados. Adicionalmente, através da solução *IntelliSuite Analytics*, a plataforma disponibiliza capacidades de análise preditiva, possibilitando a identificação antecipada de falhas e ineficiências, antes que estas originem consequências mais graves ao nível da operação e desempenho do edifício [24].

Através de um navegador web, é possível monitorizar, controlar e administrar os edifícios. A ferramenta tem custos mensais associados ao serviço [24].

A utilização da extensão *ACxelerate™ Commissioning Tool* permite realizar o comissionamento de unidades de volume de ar variável (VAV) com conduta simples, conduta dupla e ventiladores incorporados (configurações em série e em paralelo), incluindo os registos e válvulas de reaquecimento. Podem ser feitos testes funcionais que testam, de forma segura e automática, os registos VAV e/ou válvulas de reaquecimento no sistema mecânico. Para a análise dos resultados dos testes é gerado um relatório que documenta tanto o funcionamento adequado do sistema

como eventuais falhas. Este relatório pode ser exportado em formato *HTML* ou *Excel* [24].

2.5.2. EcoStruxure Building Commission

A *EcoStruxure Building Commission* é uma aplicação de uso gratuito desenvolvida pela *Schneider Electric* com o propósito de facilitar o comissionamento de equipamentos em sistemas de gestão técnica de edifícios. Esta plataforma destina-se especificamente à verificação e configuração de equipamentos *SpaceLogic BACnet/IP*, módulos de expansão dos controladores *RP* e equipamentos *EasyLogic BACnet MS/TP*, permitindo realizar estas operações sem a necessidade de usar o *EcoStruxure Building Operation (EBO)* que é uma ferramenta de gestão, monitorização e controlo de sistemas de gestão de edifícios [25].

A ferramenta possibilita a ligação direta aos controladores, o que permite aos técnicos executar testes e configurações em campo sem depender de servidores ou de licenças adicionais. A conexão aos equipamentos pode ser efetuada através de *BACnet/IP* ou por *Bluetooth*, sendo neste último caso necessário um adaptador que é conectado na lateral do controlador [25].

Entre as funcionalidades disponibilizadas pelo *EcoStruxure Building Commission* destacam-se a leitura de pontos de entrada e saída, a configuração de parâmetros operacionais e a geração automática de relatórios de verificação. Estes relatórios documentam o estado da instalação, identificando eventuais falhas ou ligações incorretas, e constituem um registo formal do progresso do comissionamento. A plataforma inclui ainda procedimentos de *I/O checkout*, que permitem confirmar a integridade das ligações físicas e a comunicação entre os controladores e os módulos de expansão. No caso dos controladores *RP*, é possível realizar testes específicos em módulos como os módulos *DALI*, onde se podem descobrir, associar e testar individualmente equipamentos de iluminação e dispositivos de entrada. Permite testar a intensidade luminosa, os módulos de estores ou persianas, nos quais se realiza a calibração e teste do movimento e os módulos de relés, destinados ao teste do acionamento das saídas. Para além destes, a aplicação suporta o comissionamento de dispositivos *Modbus* e de dispositivos *Zigbee* em sistemas com controladores *RP*, permitindo a associação destes a redes de diferentes níveis de segurança e a verificação do seu estado *online* ou *offline* [25].

Do ponto de vista prático, a utilização desta aplicação contribui para reduzir o tempo necessário para o comissionamento da obra, ao disponibilizar ferramentas rápidas de diagnóstico e ajuste. Paralelamente, diminui a dependência de técnicos altamente especializados, uma vez que tarefas de verificação inicial podem ser realizadas por técnicos com formação básica. A geração automática de documentação relativa ao processo de teste é útil para o controlo de qualidade e entrega de obra [25].

Contudo, a aplicação apresenta limitações. É mais adequada a instalações de menor complexidade, dado que a conexão é feita manualmente a um equipamento de cada vez, o que pode tornar o processo mais moroso em projetos de maior dimensão. Além disso, a compatibilidade é restrita, suportando apenas determinados equipamentos da *Schneider Electric* [25].

2.5.3. *AutoCXI*

A *AutoCXI* é uma plataforma de comissionamento automatizado para sistemas *BMS*. É desenvolvida para otimizar o processo de comissionamento e realizar testes em sistemas de gestão de edifícios. Permite integrar funcionalidades avançadas de automação e análise de resultados. A ferramenta permite gerir tanto o teste automatizado de sequências de operação como o fluxo de trabalho para testes ponto a ponto, arranque de sistemas e comissionamento. Existem custos mensais associados à subscrição do serviço [26].

Uma das características principais do *AutoCXI* é a capacidade de importar bases de dados de pontos *BACnet* a partir de plataformas *BMS* suportadas, o que permite a criação e execução de cenários de teste mais complexos. A plataforma estabelece ligação direta, seja de forma local ou através da *cloud*, com os sistemas *BMS* através do *BACnet/IP* e outras tecnologias suportadas [26].

A *AutoCXI* proporciona ainda uma documentação detalhada, com um registo do histórico completo dos testes e do comissionamento, o que permite avaliar de forma objetiva o progresso e a conformidade das operações. No domínio da criação de testes, a *AutoCXI* recorre a uma linguagem própria para a criação dos ficheiros de teste. Os testes podem ser desenvolvidos através de um editor online simples, pela reutilização de *scripts* anteriores ou pela importação de bibliotecas de testes já existentes. A plataforma disponibiliza ainda um construtor de testes online que orienta o utilizador, passo a passo, no desenvolvimento de ficheiros de teste. Existe a possibilidade de usar modelos de teste genéricos, independentes de pontos ou sistemas específicos, que podem posteriormente ser adaptados a pontos concretos. Esta flexibilidade permite que um único teste seja executado várias vezes, ao ser aplicado a diferentes equipamentos. A biblioteca padrão inclui mais de uma centena de exemplos de testes, em constante crescimento [26].

A execução dos testes é igualmente versátil. É possível programar a realização de testes para datas e horários específicos e armazenar todos os dados de cada execução numa base de dados com registo de data e hora. Os resultados são apresentados em relatórios de leitura simples, em formato de aprovação ou falha (*pass/fail*). Para além disso, a plataforma garante que, após cada execução, os pontos sobrepostos e as alterações de *setpoints* são automaticamente revertidos para o estado original [26].

No que diz respeito à apresentação de resultados, a *AutoCXI* disponibiliza modelos de relatórios síntese e detalhados. Os relatórios síntese apresentam informação básica sobre a aprovação ou falha dos equipamentos nos testes, enquanto os relatórios detalhados permitem uma análise hierárquica com possibilidade de explorar cada passo do teste e os valores reais dos pontos do *BMS* em tempo de execução. Estes relatórios podem ser personalizados com logótipos ou cabeçalhos institucionais dos clientes [26].

2.5.4. Limitações e oportunidades de inovação

A tabela 2 contém um resumo das características das ferramentas de comissionamento apresentadas nesta secção. Duas das ferramentas apresentam custos mensais associados para a sua utilização. A ferramenta *WebCTRL* é usada apenas em sistemas de ar condicionado e a ferramenta da *Schneider* apenas para

alguns equipamentos da marca. Estas duas ferramentas facilitam o processo de comissionamento, mas não são automatizadas, sendo que é necessário realizar uma verificação manual do estado e dados dos equipamentos. A ferramenta *AutoCXI* aparenta ser a ferramenta mais completa pela sua capacidade de carregar pontos *BACnet*, realizar os testes de forma autónoma e posterior criação de relatórios.

A maior parte destas ferramentas encontradas no mercado têm custos associados para a sua utilização. Outra grande desvantagem é a falta de automatização na maior parte das ferramentas, pois continua a ser necessário pessoal técnico para conferir os dados fornecidos por estas ferramentas manualmente.

Estas 3 ferramentas pretendem apresentar uma noção do *software* existente no mercado para realizar os testes de comissionamento, já que não é possível realizar uma análise a todas as existentes.

Com esta análise, foi possível obter alguns pontos úteis para este projeto, em especial, pelas características da ferramenta *AutoCXI*. Pretendeu-se desenvolver uma ferramenta com a mesma capacidade de carregar pontos *BACnet*, realizar testes a todos os equipamentos e gerar relatórios de acordo com os testes realizados. Contudo, esta ferramenta deve ser mais simples e prática, bem como adaptada às necessidades locais e com foco nos equipamentos de um fabricante em específico.

Tabela 2. Comparação das ferramentas de comissionamento.

	<i>WebCTRL</i>	<i>EcoStruxure Building Commission</i>	<i>AutoCXI</i>
Subscrição	Mensal	Gratuito	Mensal
Aplicação	Sistemas AVAC	Alguns controladores Schneider	Equipamentos <i>BACnet</i>
Relatórios	Sim	Sim	Sim
Automatizada*	Não	Não	Sim
*Carregamento de pontos e realização automática de testes			

2.6. Automatização dos testes e qualidade dos dados

Pretende-se explicar como é que a automatização pode ser aplicada no processo de comissionamento e manutenção de sistemas *BMS* e de que forma a qualidade dos dados influencia a fiabilidade e a eficiência de todo o processo.

Serão analisados quais os métodos que ainda são manuais, lentos e sujeitos a erro, que automatização é possível, embora as soluções existentes tenham lacunas, e que a qualidade e rastreabilidade dos dados é o ponto crítico do processo de digitalização do comissionamento.

2.6.1. A automatização no comissionamento

A automatização do comissionamento consiste em substituir tarefas repetitivas, lentas e suscetíveis ao erro humano por algoritmos ou ferramentas de *software* que executam testes de forma consistente e outras tarefas necessárias a esta atividade ligadas, por exemplo, ao planeamento e à implementação. A automatização não pretende eliminar a necessidade de pessoal técnico especializado, contudo pretende facilitar determinadas atividades para aumentar a produtividade e diminuir os erros.

O comissionamento é o processo de garantia de qualidade orientado para alcançar, verificar e documentar se a performance do sistema do edifício e a montagem estão de acordo com os objetivos e critérios definidos [27].

Os testes pretendem verificar se o edifício com o seu sistema está de acordo com os requisitos definidos na fase de planeamento e se segue as regras impostas pelas normas das organizações governamentais e internacionais. Estes regulamentos contemplam medidas de segurança e requisitos relacionados com a eficiência energética dos edifícios [27].

Os edifícios normalmente contêm falhas, sistemas com mau funcionamento e equipamentos não descobertos na rede durante a fase de comissionamento [27].

A base de um *BMS* é composta por *hardware*, *software* e um conjunto de protocolos de comunicação. Ultimamente, a procura pela monitorização do edifício e recolha de dados tem-se tornado notável [27].

Devido ao valor que o comissionamento de Sistemas de Gestão de Edifícios consegue agregar na garantia da sua qualidade, há uma necessidade urgente de um conjunto de ferramentas para melhorar e facilitar o processo inicial de comissionamento e também a implementação de processos contínuos e automatizados de comissionamento para permitir a deteção sistemática e efetiva de falhas [27].

Estas ferramentas visam reduzir falhas na *performance* dos edifícios, alertar sobre erros assim que estes acontecem e reduzir custos operacionais e de manutenção. Devem tratar do processo inicial de comissionamento, comissionamento de sistemas já existentes e o processo contínuo de comissionamento do edifício [27].

Em 2.2.1 viu-se quais as etapas do processo de comissionamento. Existem, essencialmente, duas etapas que podem ser otimizadas: a configuração e os testes funcionais.

É na configuração que se inicia o processo de implementação do Sistema de Gestão de edifícios. Em alguns pontos, a configuração provém do planeamento e é um processo repetitivo, pois é necessário fazer a passagem do que foi planeado para o sistema de gestão e equipamentos. A automatização deste processo passa por implementar um algoritmo capaz de, por exemplo, adaptar os ficheiros provenientes do planeamento para que possam ser criados ficheiros de configuração compatíveis a serem facilmente carregados no sistema de gestão.

Na etapa dos testes funcionais pretende-se diminuir o tempo necessário para verificar o bom funcionamento de todo o sistema. A verificação da presença de pontos no *BMS*, a leitura automática de valores e comparação com limites pré-definidos, a criação de relatórios e o envio de comandos com a análise da resposta são exemplos de atividades que podem ser automatizadas com o objetivo de reduzir, de forma significativa, o tempo necessário para os testes funcionais e o fator do erro humano.

2.6.2. Métodos e tecnologias de automatização

Nos *BMS* são usados diversos métodos e tecnologias para otimizar a gestão e o controlo dos edifícios. Para interligar as diferentes tecnologias e equipamentos, podem ser usados scripts *Python* e *Node-RED* com bibliotecas de comunicação *BACnet/IP*. Estas bibliotecas permitem integrar equipamentos *BACnet* e realizar tarefas de monitorização e controlo dos sistemas como *HVAC*, iluminação e

segurança. Já o *Node-RED* é uma plataforma de programação visual que, entre outras funções, permite automatizar e controlar equipamentos. Podem ser usados blocos específicos para *BACnet*, como o *node-red-contrib-bacnet*, a plataforma permite a criação de fluxos para gerir os equipamentos *BACnet* sem a necessidade de implementar algoritmos complexos de programação. É uma abordagem visual que permite a criação de sistema de automação predial de forma relativamente rápida [28] [29].

Para implementar diagnósticos na rede, ferramentas *open-source*, como o *BACnet Discovery Tool* e a biblioteca *BAC0* em *Python*, permitem fazer a análise e a deteção de falhas nas redes *BACnet*. O *BACnet Discovery Tool*, por exemplo, permite descobrir equipamentos e verificar a conectividade desses equipamentos nas redes *BACnet*, sendo útil para diagnóstico e manutenção em tempo real. Já a biblioteca *BAC0*, que pode ser utilizada para comunicação direta com equipamentos *BACnet* em *Python*, oferece funcionalidades avançadas para realizar testes e interagir com esses sistemas de forma automática [30] [31].

Numa instalação de um Sistema de Gestão de Edifícios, devem existir mecanismos automatizados de verificação (validação automática) para garantir que todos os equipamentos conectados (como sensores de temperatura, sistemas *HVAC* e sistemas de iluminação) operam dentro dos limites de funcionamento definidos. Se um sensor sair desses parâmetros operacionais, a lógica de validação pode disparar um alarme ou ativar ações corretivas, reduzindo riscos de falhas não detetadas que comprometam o conforto, a segurança dos ocupantes do edifício e o bom funcionamento da instalação. A fiabilidade dos sensores é fundamental para que a lógica de validação funcione corretamente. Falhas nos sensores podem comprometer o diagnóstico, sendo necessário incorporar mecanismos específicos para a deteção desses defeitos. Uma prática comum é a utilização de sensores redundantes [32].

A lógica de validação automática consiste em utilizar técnicas de deteção de anomalias e diagnóstico de falhas em tempo real, com base nos dados gerados pelos sensores do *BMS*. Por exemplo, é possível identificar sensores de *BMS* até 32 horas antes do colapso do sistema através de medidas estatísticas (como variância e autocorrelação). Além disso, abordagens baseadas em inteligência artificial para classificação e deteção de falhas têm sido aplicadas no diagnóstico automático. É possível usar algoritmos para detetar desvios nos dados adquiridos do edifício. Algoritmos de deteção de anomalias identificam estados de funcionamento inesperados, que correspondem a falhas ou a condições fora do normal, sem depender totalmente de regras pré-definidas [33], [34].

2.6.3. Qualidade dos dados

A qualidade dos dados é importante para garantir que os dados estão de acordo com os requisitos previamente estabelecidos. Contudo, é dada pouca notoriedade à análise da qualidade dos dados. Esta análise é, muitas vezes, fragmentada e limitada, com falhas nas melhores práticas e na falta do cumprimento de normas tendo em conta as necessidades específicas do edifício [35].

É necessário que exista um consenso sobre os níveis necessários da qualidade dos dados, um vocabulário comum de análise e relatório, bem como a forma de atingir a qualidade desejada [35].

A quantidade de dados recolhidos num edifício tem vindo a aumentar significativamente ao longo dos anos. Os dados estão presentes nas medições energéticas, comissionamento, calibração de simulações, entre outros. A fraca qualidade dos dados é sentida em várias áreas, desde a engenharia, às finanças e à saúde. Os dados devem estar de acordo com os requisitos da sua aplicação e os bons dados podem ajudar a tomar boas decisões e a reduzir custos, por exemplo, ao permitir um aumento da eficiência energética dos edifícios [35].

De acordo com a *International Standards Organisation* a qualidade dos dados pode ser entendida como o “grau em que um conjunto de características inerentes dos dados cumpre os requisitos” [36]. Os requisitos dos dados dependem da sua utilização, logo a qualidade dos dados deve ser definida contextualmente refletindo as necessidades e expectativas da sua aplicação [35].

A validação dos dados garante a clareza e a acessibilidade para os utilizadores dos dados e determina se um *datapoint* está em conformidade com a sua estrutura, sendo que cada *datapoint* deve ser único e livre de duplicações [35].

Os dados devem estar disponíveis num tempo razoável com níveis mínimos de desempenho, tendo em conta os requisitos da sua utilização [35].

Em caso de envio de dados para outras organizações, devem existir registos de proveniência e um registo histórico dos dados que descreva o que aconteceu com os dados durante o seu ciclo de vida (metadados) [35].

A qualidade dos dados pode ser definida a partir de oito atributos [35]:

- **Incerteza das medições:** afetada pela exatidão, precisão, fidelidade e resolução. Os dados devem ser precisos e os equipamentos de medição devem estar calibrados. A resolução enquanto menor variação na quantidade física que produz uma alteração na medição deve ser adequada à grandeza que se pretende medir. Medições com elevada resolução conseguem indicar pequenas variações nas variáveis do sistema.
- **Representatividade dos dados:** afetada por valores nulos, que causam *outliers* e são valores considerados incorretamente como nulos ou próximo de zero, e também o ruído que pode provocar instabilidade;
- **Granularidade:** a granularidade espaço-temporal é diferente da resolução. A granularidade temporal provém da frequência da medição no tempo. Já a granularidade espacial depende da grelha de medição (ex: todo o edifício, um andar ou os quartos);
- **Compleitude dos dados:** pretende garantir a qualidade temporal dos dados. A falta de informação pode resultar numa omissão de eventos importantes, como alarmes, no edifício que pode impactar na performance e até na segurança;
- **Estruturas de dados:** a conformidade com um esquema de dados é importante para assegurar a acessibilidade dos dados, a integração e interoperabilidade. Este atributo está relacionado com a forma como os dados são organizados e guardados, como em bases de dados *SQL* (*Structured Query Language*), *CSV* (*Comma Separated Values*), *XML* (*Extensible Markup Language*) e também no formato de troca de dados;
- **Tempo de acesso aos dados:** tempo entre a medição e a disponibilidade para o utilizador. Permite assegurar que os dados ainda são relevantes na altura da

sua visualização. Devem existir mecanismos de alerta para a falta de novos dados após um determinado tempo, de acordo com os requisitos.

- **Singularidade dos dados:** deve-se evitar a duplicação de dados, como a existência de diferentes equipamentos de medição com o mesmo identificador. Neste caso, se fosse um sensor de temperatura, os dados poderiam indicar duas temperaturas diferentes para o mesmo local.
- **Origem dos dados:** deve existir transparência e rastreabilidade sobre a fonte original dos dados.

A qualidade dos dados e os seus atributos podem ser representados numericamente através de medidas da qualidade dos dados. Podem ser definidos limiares para identificar áreas de fraca qualidade e para diagnosticar problemas. Estas medidas e monitorização da qualidade dos dados estão presentes na *BS ISO8000-110* [37], [35].

3. Ferramentas de automatização do planeamento e das configurações

A análise efetuada no capítulo anterior demonstrou que o comissionamento de Sistemas de Gestão de Edifícios permanece um processo dependente de tarefas manuais, com um elevado consumo de tempo e suscetibilidade ao erro humano. Embora existam ferramentas digitais de apoio no mercado, estas apresentam frequentemente limitações relacionadas ao seu custo, falta de automatização e dificuldade de adaptação a determinados trabalhos específicos.

Identificou-se a fase de configuração como uma etapa crítica, onde a implementação dos dados dos ficheiros de planeamento para o *BMS* representa um esforço repetitivo substancial. Desta forma, neste capítulo descreve-se o desenvolvimento de um conjunto de ferramentas concebidas para digitalizar este fluxo de trabalho e garantir a integridade dos dados ao longo de todo o processo

3.1. Ferramenta de Importação para o *BMS*

A primeira ferramenta desenvolvida pretendeu auxiliar na primeira etapa do comissionamento, o planeamento, e também automatizar o processo de importação dos dados do planeamento para o *BMS*. Estes dados consistem na importação dos equipamentos, a sua configuração e pontos de *software* e *hardware*.

3.1.1. Planeamento

Antes de começar o desenvolvimento da ferramenta foi necessário reunir os requisitos, como os tipos de equipamentos que existiam, a estrutura necessária que o ficheiro de importação deveria ter, os tipos de dados e outros campos adicionais.

Esta ferramenta teve como objetivo permitir a importação de dados para o *EcoStruxure Building Operation (EBO)* da *Schneider Electric*. O *EBO* suporta os seguintes tipos de equipamentos:

- *AO-V-8*: módulo com 8 saídas analógicas em tensão;
- *DI-16*: módulo com 16 entradas digitais;
- *DO-FA-12*: módulo com 12 saídas digitais com contactos simples;
- *DO-FC-8*: módulo com 8 saídas digitais;
- *UI-16*: módulo com 16 entradas universais digitais ou analógicas em tensão ou em corrente;
- *UI-8.AO-V-4*: módulo com 8 entradas universais e 4 saídas analógicas em tensão;
- *UI-8.DO-FC-4*: módulo com 8 entradas universais e 4 saídas digitais.

Ao configurar uma entrada ou saída de um equipamento no *EBO* é necessário indicar o seu tipo. Os tipos de entradas e saídas dos equipamentos pertencem à seguinte lista com as respetivas características.

- *Current Input*;
 - Escala de engenharia;
 - Escala elétrica;

- Unidade.
- *Voltage Input*;
 - Escala de engenharia;
 - Escala elétrica;
 - Unidade.
- *Resistive Input*;
 - Escala de engenharia;
 - Escala elétrica;
 - Unidade.
- *Temperature Input*;
 - Tipo de termistor (10kT1, 10kT2, 10kT3, 10kT4, 10kT5, 1k8 e 2k2).
- *CounterInput*;
 - Cor do *LED*.
- *Voltage Output*;
 - Escala de engenharia;
 - Escala elétrica;
 - Unidade.
- *Digital Output*.
 - Valor inicial.

Os pontos *BACnet* podem estar associados a uma das entradas ou saídas anteriores ou podem existir apenas em *software*. Cada ponto pode ser de um dos seguintes tipos com as respectivas características específicas de cada tipo de ponto:

- *Analog Value*;
 - *High Limit*;
 - *Low Limit*;
 - *Limit Enable*;
 - *MaxPresValue*;
 - *MinPresValue*.
- *Digital Value*;
 - *MinimumOffTime*;
 - *MinimumOnTime*.
- *Digital Input*;
- *Digital Output*;
 - *MinimumOffTime*;
 - *MinimumOnTime*.
- *Analog Input*;
 - *High Limit*;
 - *Low Limit*;
 - *Limit Enable*;
 - *MaxPresValue*;
 - *MinPresValue*.
- *Analog Output*.
 - *High Limit*;
 - *Low Limit*;
 - *Limit Enable*;
 - *MaxPresValue*;
 - *MinPresValue*.

Os seguintes campos são comuns a todos os pontos:

- Nome *BACnet*;
- Mensagem de alerta;
- Mensagem de falha;
- Mensagem de reinicialização;
- Atraso de reinicialização;
- Atraso de tempo;

O *software* usado no *BMS* permite a importação e exportação das configurações a partir de ficheiros *XML*. Para determinar qual o formato necessário para a importação, foram exportados alguns ficheiros *XML* com configurações de diferentes equipamentos em diferentes versões de *software*. A figura 16 contém um exemplo em que consta o cabeçalho do ficheiro com as informações da versão de *software* e um equipamento com a configuração de uma saída. Constatou-se que a versão de *software* não influenciava a estrutura do ficheiro *XML*, contudo, nas exportações feitas na interface *BACnet*, devia usar-se o formato *Special Export*, pois este formato continha toda a informação necessária para definir o sistema e os pontos.

```
<ObjectSet ExportMode="Special" Note="TypesFirst" SemanticsFilter="Special" Version="6.0.4.90" >
  <MetaInformation>
    <ExportMode Value="Special" />
    <SemanticsFilter Value="None" />
    <RuntimeVersion Value="6.0.4.90" />
    <SourceVersion Value="6.0.4.90" />
    <ServerFullPath Value="/Server1" />
  </MetaInformation>
  <ExportedObjects>
    <OI NAME="AO-V-8" TYPE="io.AOV8" >
      <OI NAME="Comando de velocidade" TYPE="io.point.VoltageOutput" >
        <PI Name="DefaultOutput" Unit="0x200001" Value="0" />
        <PI Name="ElecTopOfScale" Value="10" />
        <PI Name="EngBottomOfScale" Unit="0x200001" Value="0" />
        <PI Name="EngTopOfScale" Unit="0x200001" Value="100" />
        <PI Name="LogArray" Unit="0x200001" />
        <PI Name="OutputChannelNumber" Value="1" />
        <PI Name="RequestedValue" Unit="0x200001" />
        <PI Name="Value" Unit="0x200001" />
      </OI>
    </OI>
  </ExportedObjects>
</ObjectSet>
```

Figura 16. Exemplo de ficheiro de exportação de um equipamento.

Como é possível observar na figura anterior, as unidades encontram-se codificadas. Para obter os valores foi necessário extrair vários ficheiros *XML* com o conhecimento da unidade introduzida na plataforma. Assim, foi possível obter os códigos para as unidades mais usadas presentes na tabela 3. Não foram mapeadas todas as unidades existentes.

Tabela 3. Codificação das unidades no *EBO*.

Unidade	Código
m/s ²	0x20001
pH	0x500001
rad	0x30001
Nm	0x2F0001
Hz	0x120001
K	0x5F0001
Sem Unidade	0x10001
Hr	0x170001
m/s	0x310001

°C	0x280001
W	0x210001
kW	0x3210001
kVA	0x3210002
kvar	0x3220001
m ³ /h	0xF0002
m ³	0x330001
rpm	0x300002
cd	0x410001
bar	0x230006
%	0x200001

Esta informação fornece uma base para o desenvolvimento da ferramenta. Pois têm-se os elementos necessários do *EBO* que permitem criar os ficheiros *XML* importáveis no *BMS*.

3.1.2. Implementação

A ferramenta desenvolvida teve como objetivo importar dados de um ficheiro de planeamento já existente na empresa *Blin Control* e com base nessa informação criar os ficheiros necessários para a importação.

Para criar a ferramenta usou-se a linguagem de programação *Python*, tanto para desenvolver os algoritmos de leitura e conversão quanto para criar a interface gráfica. Esta linguagem já contém diversas bibliotecas que facilitam a leitura de ficheiros *Excel*, criação de ficheiros *XML* e criação de interfaces gráficas. Neste caso, foram usadas as seguintes bibliotecas:

- *Pandas*: leitura de ficheiros *Excel*, manipulação de tabelas;
- *Openpyxl*: criação de ficheiros *Excel*, edição de folhas de cálculo, leitura e escrita de células;
- *Customtkinter*: criação de interfaces gráficas;
- *Pillow*: processamento de imagens (para a interface gráfica).

Na figura 17 encontra-se a estrutura de funcionamento da ferramenta. Após a importação do ficheiro *Excel* adequado, são criados três ficheiros *XML* e um ficheiro *TGML*. Nesta estrutura existe a indicação de onde devem ser importados estes ficheiros na aplicação *EcoStruxure Building Operation*, da *Schneider Electric*. Para além disto, no Anexo A encontra-se o fluxograma com a lógica de funcionamento desta ferramenta.

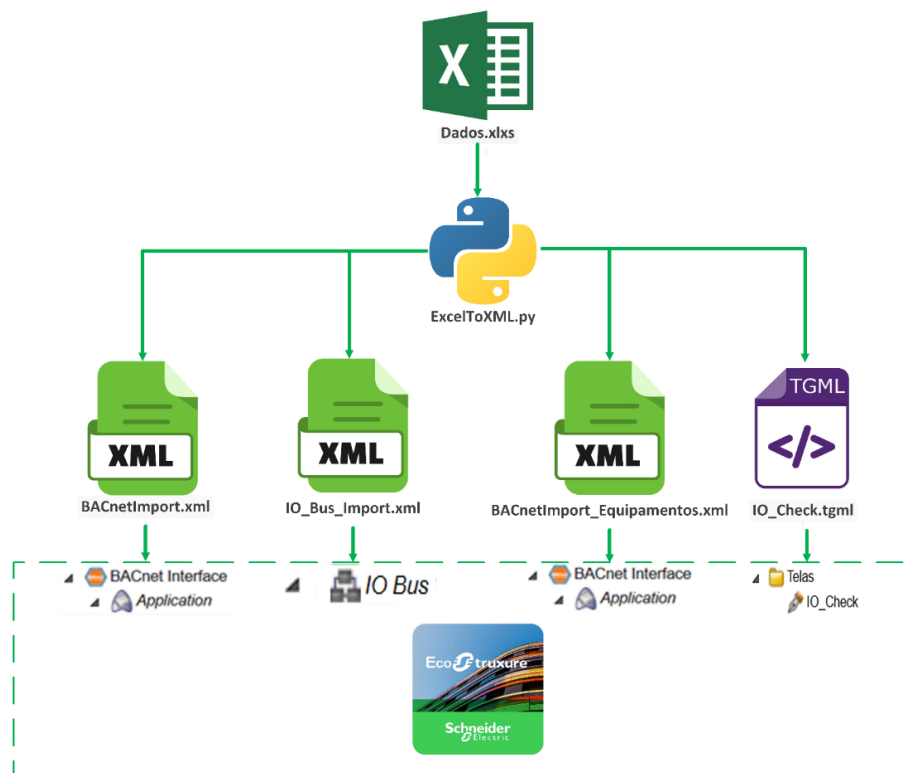


Figura 17. Fluxo de funcionamento da criação dos ficheiros para importação.

A figura 18 contém a interface gráfica criada que permite obter um ficheiro de *template* para iniciar um projeto, escolher o diretório onde está o ficheiro *Excel* a processar, escolher o diretório de saída onde devem ser guardados os ficheiros *XML* criados pela ferramenta e uma caixa de alertas para mostrar avisos e erros durante o processo de execução. Estes avisos estão relacionados a parâmetros em falta, existência de caracteres inválidos ou tipos de pontos incorretos. Os alertas podem ser críticos em que o ponto ou equipamento não é adicionado ou apenas um aviso em que o equipamento ou ponto ainda pode ser adicionado.

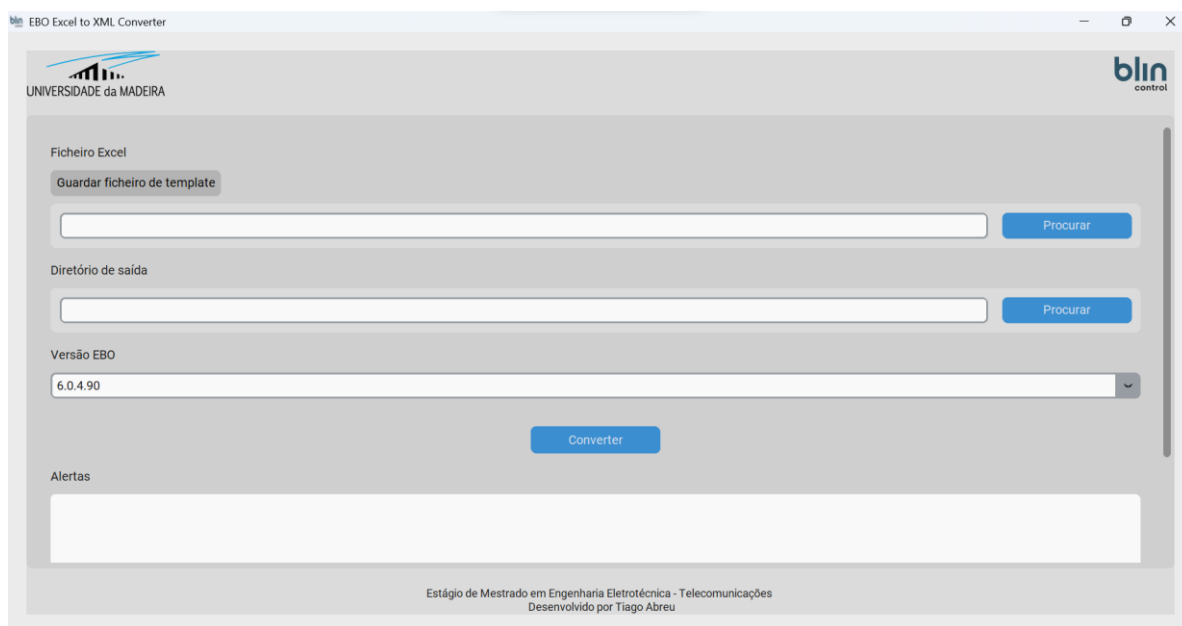


Figura 18. Ferramenta de criação dos ficheiros *XML* para importação no *EBO*.

Após iniciar a conversão, o algoritmo começa por criar os cabeçalhos dos ficheiros *XML*, como consta na figura 16. A figura 19 contém o método usado para criar os cabeçalhos do ficheiro dos equipamentos e também dos pontos *BACnet* através da função *create_bacnet_root()*, que segue a mesma lógica da função presente na figura. O outro ficheiro *XML* é criado posteriormente, pois a informação está noutra folha do *Excel*. O argumento *df* (*data frame*) contém a primeira folha do ficheiro *Excel*. Para a leitura, foi necessário indicar que duas colunas devem ser lidas no formato *string*. Estas colunas têm a informação dos identificadores numéricos dos pontos *BACnet*, contudo, cada célula tem caracteres em branco antes do identificador, através de uma regra definida no próprio *Excel*, para que o total de caracteres da célula seja igual a 8. Caso contrário, ocorre um erro ao tentar importar o ficheiro para o *EBO*.

```
def run_converter(excel_path, output_dir, ebo_version):
    global alertas_emitidos, bacnet_root, bacnet_folders
    alerta(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - O programa iniciou a execução \n")
    alertas_emitidos = False
    bacnet_root = create_bacnet_root(ebo_version)
    bacnet_folders = {}
    #equipamentos_tgml = []

    df = pd.read_excel(
        excel_path,
        dtype={
            COL_BACNET_FOREIGN_ID: str,
            COL_BACNET_FOREIGN_ID_ENABLE: str
        }
    ) #lê a primeira folha

    root = ET.Element("ObjectSet", {
        "ExportMode": "Standard",
        "Note": "TypesFirst",
        "SemanticsFilter": "Standard",
        "Version": ebo_version
    })

    #Cria secção de meta dados <MetaInformation>
    meta = ET.SubElement(root, "MetaInformation")
    ET.SubElement(meta, "ExportMode", {"Value": "Standard"})
    ET.SubElement(meta, "SemanticsFilter", {"Value": "None"})
    ET.SubElement(meta, "RuntimeVersion", {"Value": ebo_version})
    ET.SubElement(meta, "SourceVersion", {"Value": ebo_version})
    ET.SubElement(meta, "ServerFullPath", {"Value": "/Server3"})

    #Cria o espaço onde os objetos serão colocados <ExportedObjects>
    exported = ET.SubElement(root, "ExportedObjects")
```

Figura 19. Criação do cabeçalho *XML*.

Após o procedimento anterior, segue-se a criação dos equipamentos e pontos *BACnet*. A figura 20 contém o início da função que faz a leitura de uma linha do *Excel* para obter as informações da entrada/saída (E/S) de um equipamento. Primeiro são lidas as duas colunas que formam o nome da E/S e depois o tipo E/S (tipo de ponto físico). Os pontos permitidos encontram-se na página 37, no planeamento. Posteriormente, ocorre uma filtragem do nome da E/S para remover caracteres não permitidos. Os restantes campos necessários para caracterizar corretamente a E/S são obtidos a partir de um dicionário, presente no Anexo B, em que a chave do dicionário é o tipo de ponto. Neste dicionário encontram-se as características que cada tipo de ponto deve ter e a coluna do *Excel* onde é possível obter essa informação ou o valor a atribuir de forma direta, sem recorrer ao *Excel*. Em alguns casos, como as unidades, é necessário recorrer a dicionários para converter o valor lido para um valor adequado ao *XML*.

```
def create_sub_oi(parent, row, device_name):
    """Cria subelementos OI com base na linha"""
    name_a = str(row.iloc[COL_NAME_A]).strip() if not pd.isna(row.iloc[COL_NAME_A]) else ""
    name_b = str(row.iloc[COL_NAME_B]).strip() if not pd.isna(row.iloc[COL_NAME_B]) else ""
    point_type = str(row.iloc[COL_POINT_TYPE]).strip() if not pd.isna(row.iloc[COL_POINT_TYPE]) else ""

    # Ignorar linhas sem nome (A e B vazias) ou sem tipo de ponto
    if not (name_a or name_b) or not point_type:
        return

    oi_name = f"{name_a} {name_b}".strip()
    #Remover acentos, espaços extra, caracteres proibidos (:*?"<>|) e / e \ são substituídos por -
    oi_name = re.sub(
        r'[:?*"<>|]',
        '',
        f"{name_a} {name_b}"
        .replace('/', '-')
        .replace('\\', '-')
    ).strip()
    #oi_name = re.sub(r'[:?*"<>|]', '', unicodedata.normalize('NFD', f"{name_a} {name_b}".replace('/', '-').replace('\\', '-')).encode('ascii', 'ignore').decode()).strip()

    sub_oi = ET.SubElement(parent, "OI", {"NAME": oi_name, "TYPE": f"io.point.{point_type}"})

    #Cria uma lista com os campos necessários do dicionário. Se não encontrar devolve uma lista vazia.
    extras = EXTRA_FIELDS.get(f"io.point.{point_type}", [])
```

Figura 20. Criação inicial de uma entrada ou saída de um equipamento.

A figura 21 contém a configuração realizada no ficheiro *Exce/* no nome dos pontos (a) e o resultado depois de importação (b). Já a figura 22 contém as restantes colunas com as configurações que podem ser feitas no ponto. Noutras colunas é ainda indicado qual o número da entrada ou saída.

a)	
EQUIPAMENTO	DESCRIÇÃO
RVF	Falha de alimentação
CDI	Alarme de incêndio - Geral / Deslastre
DC.1	Alarme / Falta de corrente
CH.1	Alarme geral
CH.1	Falta de corrente
BPF.1	Alarme / Falta de corrente
BPQ.1	Alarme / Falta de corrente
CH.2	Alarme geral
CH.2	Falta de corrente
BPF.2	Alarme / Falta de corrente

Figura 21. Exemplo de conversão: a) Configuração no ficheiro Excel. b) Resultado da importação

IO.POINT	ESCALA ELÉTRICA		ESCALA ENGENHARIA			SENSOR TEMPERATURA	COR LED
	MIN	MAX	MIN	MAX	UN		
TemperatureInput 5	5	13	27	96	%	10kT2	Verde

Figura 22. Exemplo de configuração de pontos físicos.

Para cada ponto físico a ferramenta cria um ponto *BACnet* para representar esse ponto e um ponto *BACnet* de *enable*. Neste primeiro, podem ser adicionadas mensagens de alarme, e os respectivos limites para disparar estes eventos de alarme. A configuração base neste ponto encontra-se na figura 23 e a do ponto *enable* na figura 24. Através da configuração em XML foi possível configurar um *binding* dos pontos *BACnet* aos respectivos pontos físicos. Para criar os pontos *BACnet*, a estrutura do algoritmo é semelhante ao que foi usado para criar os pontos físicos dos equipamentos. Primeiro são preenchidos o nome e o tipo de ponto e, posteriormente, os campos extra presentes no dicionário do Anexo B que dependem do tipo de ponto *BACnet*.

PASTA	BACNET NAME	DESCRIÇÃO	TIPO	OBJECT INDEX
BAQC.1	BAQC.1 Alarme Falta de corrente	Desc22	VALUE	122

Figura 23. Preenchimento dos campos relativos ao ponto *BACnet* associado ao ponto físico.

BACNET - ENABLE - PONTO SOFTWARE			
ENABLE SUFIXO	ENABLE NAME	ENABLE DESCRIPTION	ENABLE INDEX
Alarme Falta de corrente_Ena	BAQC.1 Alarme Falta de corrente_Ena	Descricao 22	10022

Figura 24. Preenchimentos dos campos relativos ao ponto de *enable*.

No *EcoStruxure Building Operation* estes pontos ficam numa secção dedicada, uma interface *BACnet*, e são seccionados em pastas, sendo que cada pasta refere-se a um ponto físico, como demonstra a figura 27.

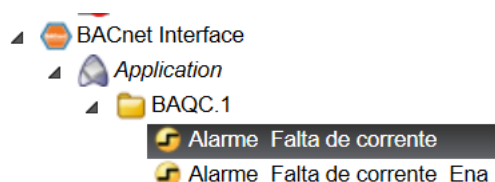


Figura 25. Resultado da importação do ficheiro relativo ao *BACnet*.

Podem ainda ser criados outros pontos *BACnet* a partir da folha “Equipamentos” presente no mesmo ficheiro *Excel*. Para cada um dos pontos, devem ser preenchidos os campos “NOME BACNET”, “TIPO” e “INDEX” como ilustra a figura 26. A indicação da pasta e descrição são de carácter opcional. Estes pontos não têm nenhuma referência (*binding*) a outros pontos.

PASTA	SUFIXO	NOME BACNET	DESCRIÇÃO	TIPO	TY	EQ	VAR	INDEX
E01	01_Enabled	E01_Enabled		BIT R/W BV	3	0 1	0 0	30100
E01	01_OperASts	E01_OperASts		BIT R BV	3	0 1	0 1	30101
E01	01_OperSts	E01_OperSts	Descrição da Zona Servida	FLOAT R AV	3	0 1	0 2	30102
E01	01_OperMd	E01_OperMd	Nome do Equipamento	FLOAT R/W AV	3	0 1	0 3	30103

Figura 26. Configuração dos pontos de *software*.

Na figura 27 está o resultado da importação a partir da configuração anterior.

Name	Value	Description	Object ID	BACnet name	BACnet type
_EqTyp	0,00		<analog-value, 30106>	E01_EqTyp	analog-value
_FilAlm	Inactive		<binary-value, 30105>	E01_FilAlm	binary-value
_FilEna	Inactive		<binary-value, 30104>	E01_FilEna	binary-value
_OperASts	Inactive		<binary-value, 30101>	E01_OperASts	binary-value
_OperMd	0,00	Nome do...	<analog-value, 30103>	E01_OperMd	analog-value
_OperSts	0,00	Descrição...	<analog-value, 30102>	E01_OperSts	analog-value

Figura 27. Importação dos pontos de *software*.

Para além dos ficheiros *XML*, a ferramenta cria um ficheiro *TGML*, também para importação no *EB0*. Este ficheiro contém a implementação de uma tela gráfica com todos os equipamentos importados e respetivas entradas e saídas. Esta tela permite facilitar a visualização do estado dos pontos de cada equipamento.

Na figura 28 encontra-se um exemplo do topo de uma tela gráfica onde consta um equipamento e os respetivos pontos. O título é preenchido com base no nome da folha do *Excel*. A zona a cinzento é destinada a mostrar o valor do ponto. Já a zona a azul é destinada a outras informações.

QEAVAC2

Sico03:DI16	DI1	CDI_Alarme		io.point.DigitalInput	
	DI2	RVF_Alarme		io.point.DigitalInput	
	DI3	Reserva_REG_1		io.point.DigitalInput	
	DI4	Reserva_REG_2		io.point.DigitalInput	
	DI5	Reserva_REG_3		io.point.DigitalInput	
	DI6	UTA3_Estado		io.point.DigitalInput	
	DI7	UTA3_Alarme		io.point.DigitalInput	
	DI8	UTA3_Filtros		io.point.DigitalInput	
	DI9	RES1_Estado		io.point.DigitalInput	
	DI10	RES1_Alarme		io.point.DigitalInput	
	DI11	VE9_Estado		io.point.DigitalInput	
	DI12	VE9_Alarme		io.point.DigitalInput	
	DI13	VE10_Estado		io.point.DigitalInput	
	DI14	VE10_Alarme		io.point.DigitalInput	
	DI15	RES2_Estado		io.point.DigitalInput	
	DI16	RES2_Alarme		io.point.DigitalInput	

Figura 28. Exemplo de tela gráfica.

Conforme o programa principal faz a leitura das linhas do *Excel*, são adicionados os equipamentos e respetivos pontos a uma lista de equipamentos da classe presente na figura 29. Quando um equipamento não existe na lista é criado e já são criadas todas as entradas e saídas tendo em conta o tipo de equipamento, conforme o dicionário presente na mesma figura. Posteriormente, são preenchidas as informações dos pontos existentes.

```
class PontoTGML:
    def __init__(self, nome_interface, tipo_interface, numero, nome_ponto="", tipo_ponto=""):
        self.nome_interface = nome_interface
        self.tipo_interface = tipo_interface
        self.numero = numero
        self.nome_ponto = nome_ponto
        self.tipo_ponto = tipo_ponto

class EquipamentoTGML:
    def __init__(self, nome, tipo):
        self.nome = nome
        self.tipo = tipo
        self.lista_pontos = criar_pontos_por_tipo(tipo)

TIPOS_EQUIPAMENTOS = {
    "DI-16": [("DI", "In", 16)],
    "DO-FC-8": [("DO", "Out", 8)],
    "UI-8.AO-V-4": [("UI", "In", 8), ("AO", "Out", 4)],
    "UI-8.DO-FC-4": [("UI", "In", 8), ("DO", "Out", 4)],
    "AO-V-8": [("AO", "Out", 8)],
    "DO-FA-12": [("DO", "Out", 12)],
    "UI-16": [("UI", "In", 16)],
}
```

Figura 29. Estrutura de dados para a criação de telas gráficas.

Terminada a leitura do *Excel* e criação dos ficheiros *XML* é criado o ficheiro da tela gráfica. À semelhança desses ficheiros, primeiro é criado o cabeçalho do ficheiro *TGML*, para então começar a criação da restante estrutura tendo em conta os dados

que foram armazenados na lista de equipamentos. Existem, essencialmente, três partes que compõem o ficheiro *TGML*: cabeçalho, caracterização do equipamento e caracterização de uma linha.

Um dos desafios que surgiram durante a implementação desta funcionalidade foi o cálculo dinâmico dos espaçamentos e tamanho das tabelas, dado que nem todos os equipamentos têm o mesmo número de entradas e saídas. Na figura 30 encontram-se os cálculos que são feitos para definir as diferentes medidas necessárias para que a tabela de um equipamento seja criada de forma correta. Cada linha da tabela tem uma altura de 20 *pixels*. As restantes medidas foram obtidas através de comparação de ficheiros já existentes na empresa.

```
def gerar Equipamento(eq, top_base):  
    linhas = []  
    top = 0  
  
    for p in eq.lista_pontos:  
        linhas.append(gerar_linha_io(p, top))  
        top += 20  
    linhas_io = "\n".join(linhas)  
    altura = top  
    offset = altura/2 - 12.5  
    meio = altura/2  
    return f'''  
<Component Clip="False" ContentHeight="{altura}" ContentWidth="970.0"  
    Height="{altura}" Id="{eq.tipo}" Left="20.0" Name="{eq.nome.replace(":", "")}" Top="{top_base}" Width="970.0">  
  
    <Rectangle Fill="None" Height="{altura}" Left="0.0" Opacity="1.0"  
        RadiusX="0.0" RadiusY="0.0" Stroke="#006393"  
        StrokeDashArray="0.0" StrokeWidth="5.0" Top="0.0" Width="970.0"/>  
  
    <Component Clip="False"  
        ContentHeight="25.0"  
        ContentWidth="{altura}"  
        Height="25.0"  
        Left="{offset}"  
        Top="{offset}"  
        Width="{altura}">
```

Figura 30. Topo da função que cria a tabela de cada equipamento em *TGML*.

Após testar e verificar o bom funcionamento da ferramenta, fez-se um manual de instalação e utilização.

Em suma, o desenvolvimento desta ferramenta permitiu automatizar a transição entre a fase de planeamento e a configuração no *EBO*. Através da criação automática de ficheiros *XML* e *TGML*, assegurou-se não só uma maior rapidez no processo, como também a uniformização das configurações, reduzindo drasticamente a probabilidade de erros associados à introdução manual dos dados do planeamento no *EBO*. Esta digitalização garante que a estrutura do sistema no *BMS* reflete o planeamento inicial, promovendo a integridade da informação desde o início do projeto. Contudo, nos projetos de automação de edifícios ocorre, frequentemente, a existência de múltiplas revisões dos ficheiros de planeamento ao longo do tempo. Nesse sentido, e como complemento necessário à ferramenta de importação, a secção seguinte descreve o desenvolvimento de uma Ferramenta de Comparação Automática de Versões, concebida para identificar diferenças entre ficheiros de listas de pontos sistemática e eficiente.

3.2. Ferramenta de comparação automática de versões

Embora a ferramenta de importação tenha otimizado a transição de dados do ficheiro de planeamento para o *BMS*, muitas vezes é necessário realizar revisões nos ficheiros de planeamento. A verificação manual destas alterações é morosa e muitas diferenças podem passar despercebidas. Logo, surge a necessidade de uma solução que automatize esta análise comparativa, garantindo maior rigor e rastreabilidade ao longo do desenvolvimento. Assim, tendo como base a ferramenta anterior, pretendeu-se implementar um programa que comparasse versões de ficheiros *Excel* de planeamento e apresentasse as diferenças entre eles. O objetivo foi permitir a deteção rápida de alterações realizadas entre ficheiros de uma mesma instalação. Pretendeu-se:

1. Verificar se todos os equipamentos existem em ambos os ficheiros;
2. Verificar, nos equipamentos existentes e em ambos os ficheiros, os pontos e suas características nas três listas:
 - a) Lista de pontos físicos;
 - b) Lista de pontos *BACnet* associados aos pontos físicos;
 - c) Lista de pontos *BACnet Enable*.

Para verificar se um equipamento ou ponto não existe numa das versões, é usado o nome do equipamento. Caso um equipamento exista, devem ser comparadas as suas características em ambas as versões. Para verificar a existência de um ponto e detetar alterações de nome, é usada a linha do ficheiro *Excel*.

Para iniciar a implementação, começou-se por usar o código desenvolvido anteriormente, pois este já permitia fazer a leitura dos pontos e equipamentos no ficheiro *Excel*. Removeu-se o código relativo aos ficheiros *XML* e *TGML*, deixando apenas a leitura do ficheiro. Na interface gráfica também foram removidos alguns elementos e adicionou-se um segundo elemento para carregar outro ficheiro, sendo que o resultado obtido foi o da figura 31.

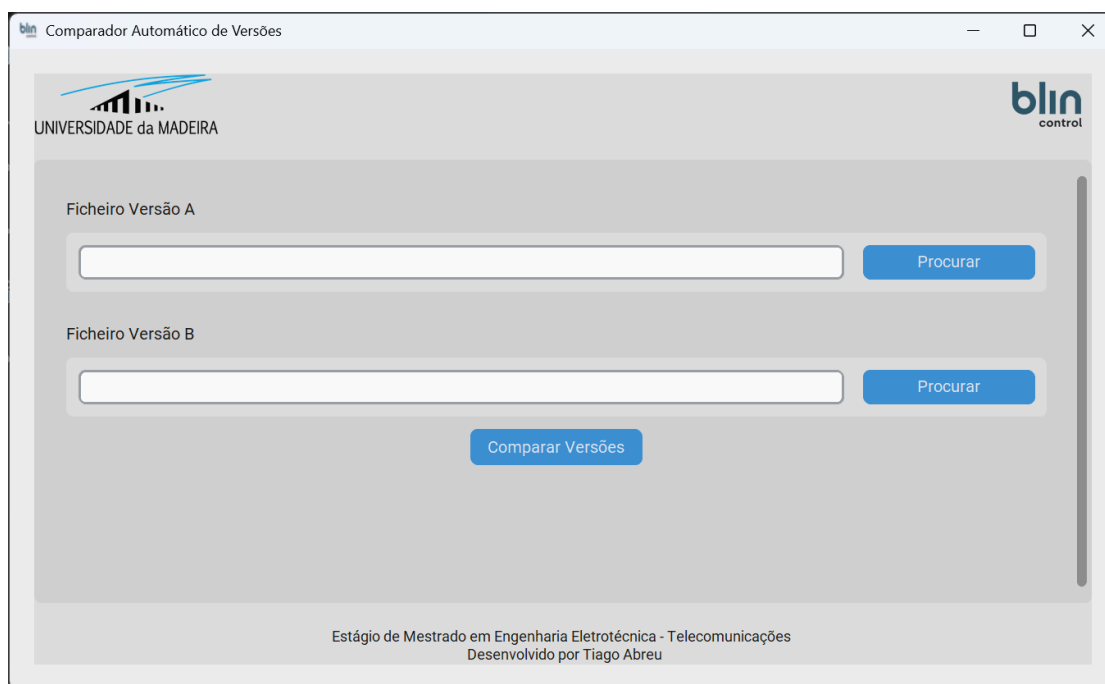


Figura 31. Interface gráfica do *software* de comparação de versões.

Posteriormente, foram criadas as classes presentes na figura 32 com o objetivo de guardar o conteúdo de cada ficheiro numa lista de equipamentos com as listas de pontos para comparação. A cada equipamento e ponto, é atribuído um campo de mensagem e de *status*, para guardar os resultados da comparação.

a)

```
class Equipamento:
    def __init__(self, nome, identificador):
        self.nome = nome
        #self.tipo = tipo
        self.lista_pontos = [] #PT
        self.lista_pontos_bacnet = [] #BC
        self.lista_pontos_enable = [] #EN
        self.identificador = identificador #
```

b)

```
class Ponto:
    def __init__(self, nome_ponto="", tipo_ponto="", descricao="", index="", row=None):
        self.nome = nome_ponto
        self.tipo = tipo_ponto
        self.descricao = descricao
        self.index = index #BACnet index
        self.row = row #linha do excel
```

Figura 32. Estruturas de dados para comparação de ficheiros: a) classe de equipamentos b) classe de pontos.

A figura 33 contém as primeiras etapas para a comparação. Após carregar a informação dos ficheiros para as listas de equipamentos, as listas são transformadas em dicionários indexáveis por nome, por facilitarem a comparação. A primeira comparação ocorre nos equipamentos, em que são verificados os que existem em ambas as listas e os que apenas existem na primeira ou na segunda lista. Posteriormente, são guardadas as mensagens de erro e o *status* de cada equipamento conforme a informação presente em cada dicionário.

```
equipamentos1 = []
equipamentos2 = []
equipamentos1 = run_converter(excel_path)
equipamentos2 = run_converter(excel_path2)

#Transformar as listas de equipamentos em dicionários
dict1 = sct.indexar_equipamentos(equipamentos1)
dict2 = sct.indexar_equipamentos(equipamentos2)

#Verificar que equipamentos são comuns nos dicionários
nomes1 = set(dict1.keys())
nomes2 = set(dict2.keys())
apenas_1 = nomes1 - nomes2
apenas_2 = nomes2 - nomes1
comuns = nomes1 & nomes2
```

Figura 33. Código para comparação de equipamentos.

Para os equipamentos presentes em ambas as listas, são comparados os pontos desses equipamentos. A figura 34 contém o código inicial para a comparação da lista de pontos. A função que faz a comparação retorna uma estrutura de dados que contém a informação sobre pontos com características diferentes ou pontos que existem apenas num ficheiro. Esta informação é usada posteriormente para criar o relatório. No Anexo C encontra-se o código de comparação. As listas de pontos são convertidas em dicionários indexáveis pela linha *Excel*. Os parâmetros a comparar são o nome do ponto, descrição e endereço *BACnet*, se aplicável.

```
for nome in comuns: #Verificação dos pontos existentes em ambos os equipamentos
    eq1 = dict1[nome]
    eq2 = dict2[nome]

    eq_res = sct.ResultadoEquipamento(nome, eq1.identificador)
    eq_res.status = "ok"
    eq_res.mensagens.append("Equipamento presente em ambas as versões")

    pontos_fisicos = sct.comparar_listas_pontos(eq1.lista_pontos, eq2.lista_pontos, "PT")
    pontos_bacnet = sct.comparar_listas_pontos(eq1.lista_pontos_bacnet, eq2.lista_pontos_bacnet, "BC")
    pontos_enable = sct.comparar_listas_pontos(eq1.lista_pontos_enable, eq2.lista_pontos_enable, "EN")
```

Figura 34. Código inicial de comparação das listas de pontos.

Terminada a análise, o programa mostra uma janela com um relatório contendo os resultados das diferenças entre os dois ficheiros carregados inicialmente. Um exemplo deste relatório encontra-se na janela da figura 35. Nela estão evidenciados exemplos de pontos diferentes e alterações entre as versões



Figura 35. Relatório de comparação de versões.

Esta ferramenta foi concebida para fornecer um mecanismo rigoroso e eficiente para a deteção de disparidades entre versões de um projeto. Ao permitir a identificação sistemática de alterações em equipamentos e pontos, como modificações nos nomes, descrições ou identificadores *BACnet*, o programa elimina a falibilidade da verificação manual e assegura a integridade dos dados ao longo de todo o ciclo de desenvolvimento do projeto. Esta solução consolida a qualidade da informação que sustenta o sistema de gestão.

No capítulo seguinte será apresentada a Ferramenta de Testes, a componente final deste conjunto de aplicações, concebida para interagir diretamente com os controladores para realizar a monitorização e os testes funcionais necessários para assegurar a conformidade operacional dos edifícios

4. Ferramenta de Testes

Após assegurar a integridade da informação através da ferramenta de comparação de versões, passou-se para a fase de testes funcionais e validação final do comissionamento. O presente capítulo descreve o desenvolvimento da ferramenta concebida para interagir diretamente com a infraestrutura física do edifício, para garantir que os equipamentos operam conforme o esperado. Esta ferramenta teve como objetivo usar os pontos *BACnet* para implementar testes nos equipamentos das instalações para reduzir o tempo das manutenções realizadas pelos técnicos e também para reduzir o erro humano associado à realização manual deste tipo de atividades.

A escolha do protocolo *BACnet* para o desenvolvimento da ferramenta apresentada justifica-se, prioritariamente, pela sua utilização por parte da Blin Control, empresa onde decorreu o estágio, uma vez que esta utiliza o *BACnet/IP* como padrão central nas suas implementações. Do ponto de vista técnico e de arquitetura, este protocolo é reconhecido como a solução que melhor permite a interligação entre a camada de gestão e a camada de automação. Sendo um protocolo aberto e normalizado internacionalmente pela *ASHRAE*, o protocolo assegura a interoperabilidade entre equipamentos de diferentes fabricantes, o que facilita a integração de diversos sistemas num ambiente comum.

4.1. Bibliotecas *BACnet*

Primeiramente, foi necessário avaliar as bibliotecas *BACnet* existentes para a linguagem de programação *Python* para, posteriormente, realizar a implementação da ferramenta.

A biblioteca escolhida deveria permitir:

- Ler e escrever valores nos pontos *BACnet*;
- Descobrir equipamentos na rede e suas características;
- Suportar vários tipos de objetos *BACnet*;
- Simplificar a implementação.

A partir destas características procuraram-se as bibliotecas mais conhecidas a fim de escolher a mais adequada a este trabalho.

A primeira biblioteca analisada foi a *BACpypes3*. Esta biblioteca implementa o protocolo *BACnet* de forma completa num nível mais baixo de arquitetura. Permite a criação de servidores, clientes e equipamentos virtuais, bem como acesso a objetos complexos. Contudo, é uma biblioteca complexa de usar com uma configuração também complexa e documentação quase inexistente. Na documentação é recomendado desligar a *firewall* do equipamento em que a biblioteca é usada ou permitir a passagem de pacotes na porta 47808 [38].

A segunda biblioteca analisada, *BAC0*, tem como base a primeira biblioteca e simplifica as tarefas mais comuns. O seu foco está em facilitar a utilização do protocolo por programadores de Sistemas de Automação de Edifícios. Permite a leitura, a escrita e suporta diversos objetos *BACnet*. Contudo, não permite objetos do tipo *file*, *program*, *notification class* e *structured objects*, o que não é problemático para a implementação

que se pretendeu fazer. Ao usar esta biblioteca, o computador passa a funcionar como um equipamento *BACnet*. Podem ser criados vários equipamentos, desde que sejam usadas interfaces (portas) diferentes [31].

Foram ainda analisadas outras duas bibliotecas, *pyBACnet* e *BACnetPY*. A primeira usa executáveis de uma *stack BACnet* em linguagem C, sendo pouco desenvolvida e apenas com funções de leitura e escrita de objetos [39]. A segunda não cobre objetos complexos e a sua documentação é mínima, sendo mais usada para pequenos testes [40].

Tendo em conta as características mencionadas, optou-se por usar a biblioteca *BAC0*, sendo que, começou-se por efetuar alguns testes básicos com esta biblioteca.

No Anexo D encontra-se um código criado para encontrar equipamentos *BACnet* na rede, fazer a leitura de alguns parâmetros num equipamento conhecido e escrita de um valor digital numa determinada prioridade. Neste caso, a procura foi feita pelo endereço *IP* do equipamento, mas pode ser realizada pelo seu nome ou endereço *BACnet*. No caso da escrita, para remover o valor escrito anteriormente numa prioridade, basta escrever “*null*” no campo do valor para essa prioridade.

A biblioteca *BAC0* encontra os equipamentos através do envio de mensagens de *broadcast* na rede. O protocolo *BACnet* define um mecanismo de descoberta baseado em mensagens *Who-Is* / *I-Am*. Este processo ocorre da seguinte forma:

1. Um cliente *BACnet* envia uma mensagem *Who-Is* para a rede.
2. Essa mensagem pode ser enviada em *broadcast*, permitindo que todos os dispositivos *BACnet* na rede a recebam.
3. Cada equipamento *BACnet* que recebe o pedido *Who-Is* responde com uma mensagem *I-Am*.
4. A mensagem *I-Am* contém informações como:
 - *Device Identifier* (identificador único do equipamento);
 - Endereço de rede;
 - *Vendor ID*;
 - *Capabilities* do equipamento, uma lista objetos e propriedades.

Desta forma, o cliente que inicia a descoberta consegue identificar todos os equipamentos ativos na rede *BACnet*. A biblioteca *BAC0* simplifica a interação com o protocolo *BACnet* ao abstrair grande parte da complexidade. Após o envio do pedido de descoberta, o cliente aguarda durante um intervalo de tempo configurado pelas respostas dos equipamentos. Cada resposta *I-Am* recebida é processada e adicionada à lista de equipamentos conhecidos.

A biblioteca *BAC0*, pelos testes realizados, apresentou diversas vantagens para este projeto. Permitiu uma abstração à complexidade do protocolo *BACnet*, fornecendo uma integração simples com a linguagem de programação *Python* e leitura dos objetos *BACnet* após a descoberta dos equipamentos, entre outras funcionalidades que não serão usadas neste trabalho, mas que em caso de necessidade futura podem ser implementadas na ferramenta desenvolvida.

4.2. Planeamento

Começou-se por definir um modelo de ficheiro *Excel* para que sejam indicadas as configurações dos equipamentos e pontos *BACnet* a testar. O ficheiro com as configurações dos testes a realizar tem o cabeçalho da figura 36.

EQUIPAMENTO			PONTOS			TESTE DE LIMITES					TESTES SEQUENCIAIS
NOME BACnet	ENDEREÇO IP	BACnet ID	NOME	TIPO	BACnet ID	LOW LIMIT	HIGH LIMIT	UNIDADE	CATEGORIA	SUBCATEGORIA	

Figura 36. Cabeçalho do ficheiro de configuração dos testes.

No equipamento, apenas é necessário preencher o seu nome *BACnet*. Caso seja indicado o seu endereço *IP* ou *BACnet ID*, estes devem ser validados pela ferramenta e deve ser emitido um alerta caso não correspondam aos valores presentes nos equipamentos da rede.

Caso existam pontos com o mesmo identificador numérico (*BACnetID*), devem ser indicados os tipos de pontos para que possam ser distinguidos. Caso contrário, pode ser indicado apenas o nome do ponto ou o seu identificador numérico.

O Teste de Limites é destinado aos pontos do tipo *analog-input* e *analog-value*, caso os campos do teste estejam preenchidos. Ao selecionar a categoria e a subcategoria os parâmetros dos limites e unidade são automaticamente preenchidos. Estes parâmetros encontram-se na tabela do Anexo E.

O funcionamento dos Testes Sequenciais será explicado numa secção dedicada.

Na figura 37 encontra-se um exemplo de preenchimento do ficheiro.

EQUIPAMENTO			PONTOS			TESTE DE LIMITES					TESTES SEQUENCIAIS
NOME BACnet	ENDEREÇO IP	BACnet ID	NOME	TIPO	BACnet ID	LOW LIMIT	HIGH LIMIT	UNIDADE	CATEGORIA	SUBCATEGORIA	
RP-C-16A_TestBACnet			Resistive Input	analogInput		0	10000	ohms	Resistência		1:Read:+
RP-C-16A_TestBACnet			Analog Value	analogValue		49	51	hertz	Frequência	Rede	1:Read:-;2:Read:+
RP-C-16A_TestBACnet			Digital Input	binaryInput	55						1:Read:-
RP-C-16A_TestBACnet			c								1:Write;2:Write
RP-C-16A_TestBACnet			Digital Output								1:Write
RP-C-16A_TestBACnet					59	-1	1	no-units			1:Write;2:Write
RP-C-16A_TestBACnet					60	-1	1	no-units			1:Write;2:Write
RP-C-16A_TestBACnet					61	0	120	percent	Capacidade		1:Write;3:Write
RP-C-16A_TestBACnet					62	215	245	volts	Tensão	Simple	1:Write;3:Write
RP-C-16A_TestBACnet					63	0	3000	amperes	Corrente	Nominal	1:Read:++;2:Read:-
RP-C-16A_TestBACnet					64						
RP-C-16A_TestBACnet					65						
RP-C-16A_TestBACnet					66	0	120	percent	Capacidade		
RP-C-16A_TestBACnet					67	49	51	hertz	Frequência	Rede	

Figura 37. Exemplo de configuração do ficheiro de equipamentos para testes.

Para além do Teste de Limites e Testes Sequenciais referidos anteriormente, pretendeu-se implementar ainda:

- Teste de presença, para verificar se todos os equipamentos e pontos do ficheiro são encontrados na rede;
- Teste nas entradas digitais para detetar mudanças de estado;
- Teste nas saídas digitais e analógicas para permitir alterar os valores de saída e verificar fisicamente o impacto dessas alterações;
- Amostragem de dados e representação gráfica dos mesmos;
- Criação automática de relatórios após a realização de cada teste;
- Permitir o funcionamento do programa em ambiente *Windows* e *Linux*.

4.3. Implementação

Após os testes feitos com a biblioteca passou-se ao desenvolvimento da ferramenta de testes. Para criar a interface gráfica usou-se a mesma biblioteca da primeira ferramenta, *CustomTkinter*.

A figura 38 apresenta a arquitetura geral da ferramenta de testes desenvolvida. O processo inicia-se com a leitura de um ficheiro Excel que contém informação sobre os testes a realizar e a identificação dos pontos *BACnet* a testar. A aplicação utiliza esta informação para estabelecer comunicação com os equipamentos através do protocolo *BACnet/IP*. Durante a execução dos testes são efetuadas operações de leitura e escrita sobre os pontos *BACnet*, o que permite validar o funcionamento dos equipamentos. No fim de cada teste, os resultados obtidos são organizados e convertidos automaticamente num relatório em formato *PDF*.

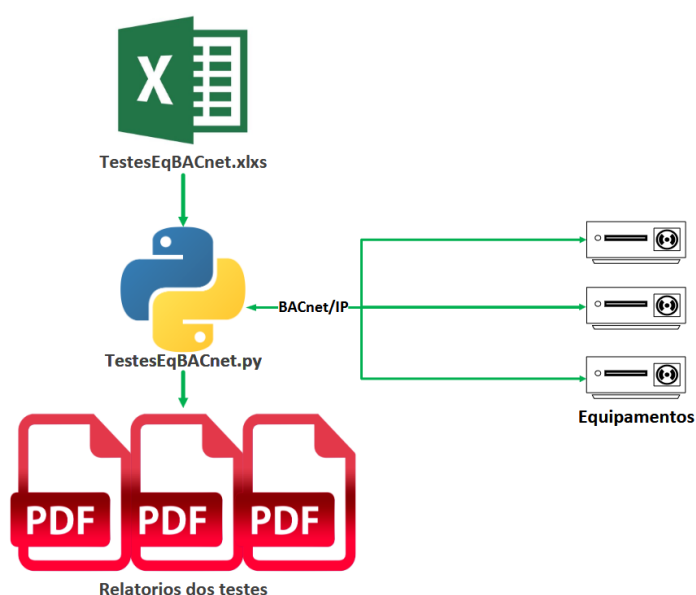


Figura 38. Fluxo de funcionamento da ferramenta de testes funcionais.

No Anexo F encontra-se a lógica de funcionamento do programa, que será descrita ao longo desta secção.

O primeiro passo foi permitir a configuração da interface *BACnet* e os parâmetros de rede. A janela da figura 39, permite escolher qual o endereço *IP* e máscara de rede adequados, a porta a usar nas comunicações (por padrão 47808₁₀, BAC0₁₆) e o endereço *BACnet* para o equipamento que está a executar o programa. Este endereço deve ser diferente dos endereços dos restantes equipamentos na rede.

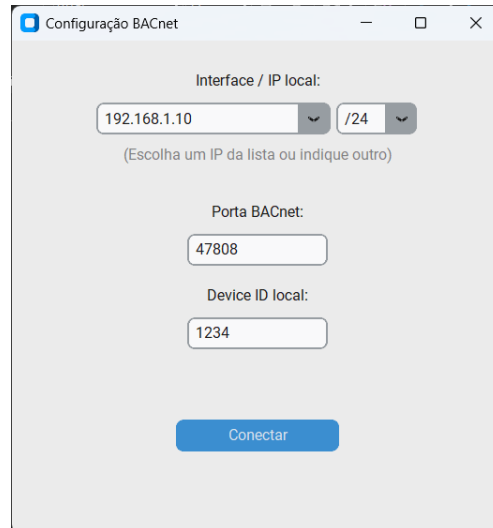


Figura 39. Interface das configurações *BACnet*.

O código apresentado na figura 40 é responsável pela inicialização da interface *BACnet* da aplicação, através de uma arquitetura assíncrona baseada em *asyncio* e *threading*. Esta abordagem permitiu separar a comunicação *BACnet* da interface gráfica, para garantir que a aplicação permanecesse responsiva durante leituras, escritas e restantes operações de rede.

Inicialmente, a função `_abrir_config_bacnet()` abre a janela de configuração mostrada acima onde o técnico introduz os parâmetros necessários, nomeadamente o endereço *IP*, a porta e o identificador do equipamento (*device_id*). Após a confirmação dos dados, estes são enviados para a função `_iniciar_bacnet()`. Nesta função é criada uma tarefa através da biblioteca *threading*. Isto evita que as operações *BACnet* bloqueiem o ciclo principal da interface gráfica, permitindo que a aplicação continue a responder normalmente às ações do utilizador. Dentro da tarefa é criado um ciclo de eventos assíncronos (*event loop*) recorrendo à biblioteca *asyncio*, através das funções `asyncio.new_event_loop()` e `asyncio.set_event_loop()`. Este ciclo de eventos é responsável pela execução de tarefas assíncronas (*async/await*) relacionadas com a comunicação *BACnet*. Posteriormente ocorre a inicialização da *stack BACnet* através da biblioteca *BAC0*. Nesta fase são configurados o endereço *IP* local, a porta *BACnet* e o identificador *BACnet* do equipamento criado pela aplicação. Por fim, o método `run_forever()` mantém o ciclo assíncrono em execução contínua durante todo o funcionamento da aplicação. Desta forma, todas as operações *BACnet*, como leituras e escritas, podem ser executadas de forma paralela ao funcionamento da interface gráfica e outras tarefas que possam estar em execução simultaneamente.

Ao longo do desenvolvimento, pretendeu-se separar a lógica de comunicação *BACnet* da interface gráfica, para melhorar a responsividade do sistema e garantir maior estabilidade durante operações.

```

def _abrir_config_bacnet(self):
    ConfigBACnet(self, self._iniciar_bacnet)
def _iniciar_bacnet(self, ip, porta, device_id):
    def runner():
        self.loop = asyncio.new_event_loop()
        asyncio.set_event_loop(self.loop)

        def start():
            self.bacnet = BAC0.start(
                ip=ip,
                port=porta,
                deviceId=device_id
            )

        self.loop.call_soon_threadsafe(start)
        self.loop.run_forever()

    threading.Thread(target=runner, daemon=True).start()

```

Figura 40. Configuração da interface *BACnet*.

Terminada a configuração inicial, a próxima janela permite importar as configurações dos equipamentos e pontos a testar a partir do ficheiro *Excel* da secção anterior. Após carregar os dados, a ferramenta mostra todos os equipamentos e pontos lidos. Também é possível visualizar o número de equipamentos e pontos carregados. Um exemplo de importação encontra-se no Anexo G. Estes equipamentos e pontos são carregados para uma lista de equipamentos com uma estrutura de dados semelhante às classes da figura 32. À classe de pontos foram adicionados parâmetros auxiliares que permitem guardar a informação dos testes realizados, como resultado de testes, mensagens de erro e registo de prioridades onde foi efetuada a escrita para posterior limpeza. Estes parâmetros encontram-se na figura 41.

```

class Ponto:
    def __init__(self, nome, tipo=None, bac_id=None):
        self.nome = nome
        self.tipo = tipo
        self.bac_id = bac_id
        self.status = "ok" # ok | alterado | erro
        self.mensagens = [] # lista de mensagens
        #Teste de Limites
        self.low_limit = None # Limite inferior para teste analógico de entrada
        self.high_limit = None # Limite superior para teste analógico de entrada
        self.unit = None # Unidade para o teste analógico de entrada
        #--
        self.estado_teste = None # Resultado do teste (ok / erro / alterado / etc.) (pendente / ok / fail)
        self.mensagem_teste = "" # Mensagem adicional do teste
        self.present_value = None # Último valor lido do ponto
        self.valor_inicial = None
        self.prioridade = None #Prioridade de escrita de valores
        self.prioridades_escritas = [] #Lista com os ids das prioridades escritas durante os testes
        #Teste de pares
        self.tp_lista_testes = [] #Lista de testes para o teste de pares
        #Teste Sequencial
        self.teste_sequencial = False
        self.teste_sequencial_tipo = None #w - write ou r - read
        self.teste_sequencial_resultado = None # "+" ou "-"
        self.limites_min = None
        self.limites_max = None
        self.limites_inicial = None

```

Figura 41. Estrutura de dados para guardar a informação dos pontos na ferramenta de testes.

Terminado o carregamento é possível realizar o primeiro teste que pretende verificar se os equipamentos e pontos carregados estão na rede ao carregar em “Validar Rede *BACnet*”. Na interface gráfica é possível escolher um tempo de *timeout* para a procura dos equipamentos que foi pré-definido para 5 segundos. Em redes muito grandes pode ser necessário aumentar este tempo. O programa começa por procurar os equipamentos existentes na rede, de acordo com a lógica apresentada na figura 42.

Após *timeout*, se não forem recebidas respostas de novos equipamentos, o que equivale a não encontrar novos equipamentos, a procura termina. Se não for encontrado nenhum equipamento na rede, o programa não passa para a comparação dos equipamentos carregados com os equipamentos descobertos. Caso existam equipamentos, é feita a procura dos equipamentos carregados na lista de equipamentos encontrados na rede, bem como dos pontos de cada equipamento. Na secção 4.6.1, será vista uma correção implementada para melhorar a procura de pontos diferentes com o mesmo endereço *BACnet*.

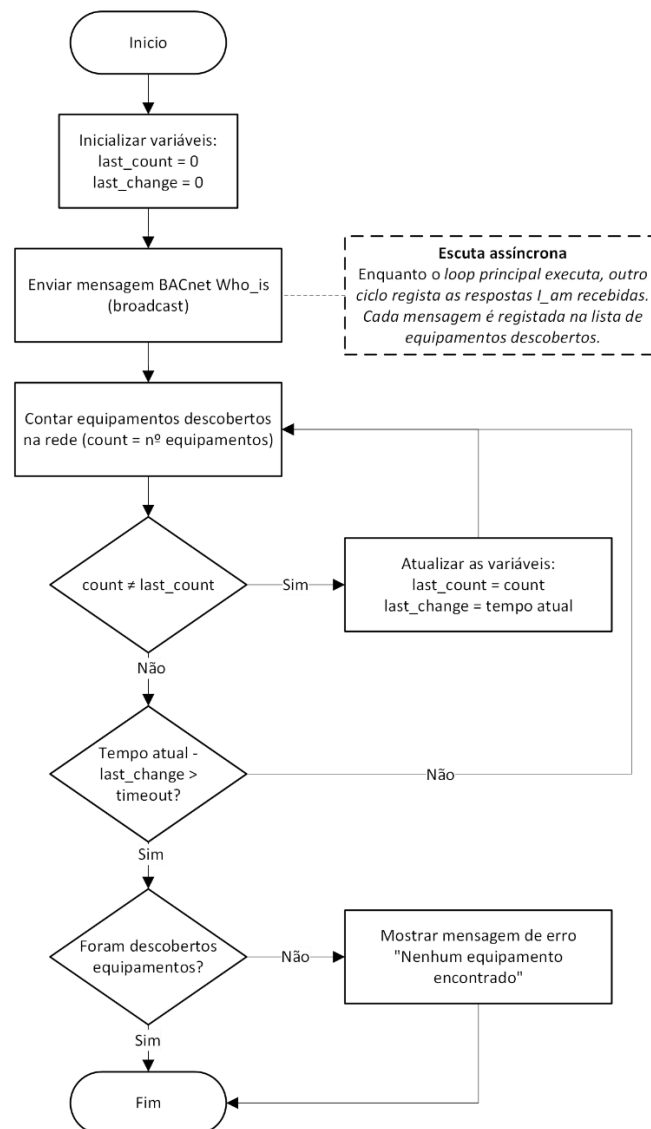


Figura 42. Fluxograma do processo de descoberta de equipamentos BACnet na rede.

Na figura 43 é possível visualizar os possíveis erros e alertas que podem surgir ao realizar a validação.

Feita a validação devem aparecer dois botões na parte inferior da interface gráfica. Um botão permite abrir uma nova janela para os próximos testes e outro permite guardar um relatório relacionado ao teste de presença.

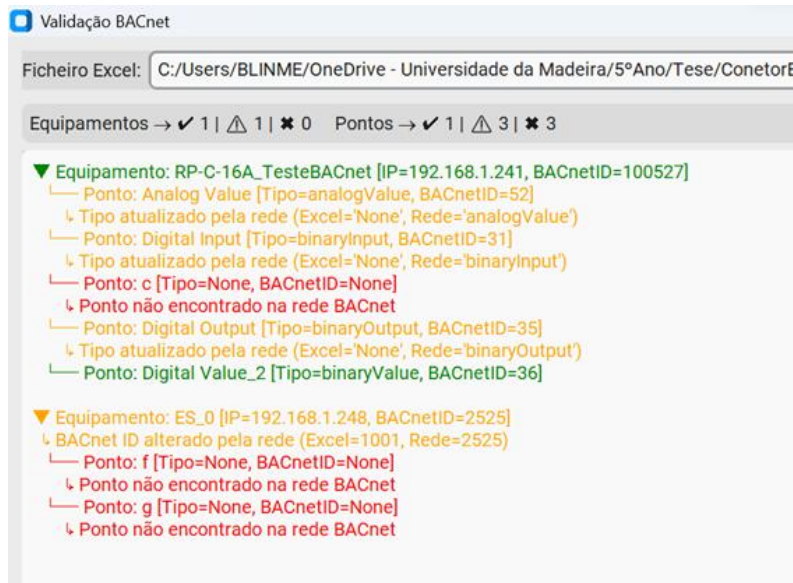


Figura 43 Validação dos equipamentos e pontos *BACnet*.

Ao passar para a nova janela para realizar os restantes testes, ocorre uma filtragem em que apenas são usados os equipamentos e pontos sem erros (cor verde ou amarela). A função responsável pela filtragem percorre a lista de equipamentos e remove aqueles que têm o *status* de erro. Os equipamentos e pontos com alertas (cor amarela) ainda são mantidos para testes com os valores atualizados para aqueles que foram encontrados na rede. No Anexo H encontra-se uma figura com a janela de testes. No canto superior esquerdo existe ainda um botão que permite guardar o relatório do último teste realizado e outro que permite editar as mensagens de erro/alerta presentes nos pontos.

A escolha dos testes implementados nesta ferramenta foi efetuada tendo em consideração as necessidades identificadas durante o estágio na empresa Blin Control, em que se procurou automatizar os procedimentos do comissionamento que eram, até então, realizados manualmente pelos técnicos. Foram privilegiados os testes que representam as verificações mais frequentes efetuadas durante o processo de comissionamento.

4.3.1. Testes nas entradas analógicas

Este teste permite verificar se os pontos do tipo *analog-value* e *analog-input* estão dentro dos parâmetros definidos no ficheiro importado na ferramenta, bem como verificar as unidades.

O código apresentado na figura 44 faz parte de um ciclo que percorre a lista de pontos e realiza a leitura e validação dos limites desses pontos *BACnet*. O objetivo principal é verificar se o valor atual do ponto (*presentValue*) e a respetiva unidade (*units*) correspondem aos parâmetros esperados que foram definidos previamente. Inicialmente são efetuadas duas leituras no equipamento de destino. A primeira leitura obtém o valor atual do ponto (*presentValue*), enquanto a segunda obtém a unidade associada ao objeto *BACnet* (*units*). O código verifica primeiro se o valor recebido é numérico e se não corresponde a um valor inválido. Caso o valor seja válido, é efetuada uma comparação com os limites mínimo (*low_limit*) e máximo (*high_limit*) definidos para o ponto. Sempre que o valor esteja fora do intervalo permitido, é

adicionada uma mensagem de erro à lista de mensagens, indicando que o valor está fora dos limites aceitáveis. Caso o valor lido não seja numérico ou seja inválido, é adicionada uma mensagem a indicar que o *presentValue* é inválido. De seguida o código compara a unidade esperada (*pt.unit*), com a unidade efetivamente lida (*units*). Se as unidades não coincidirem, é adicionada uma mensagem de erro à lista de mensagens, indicando a unidade esperada e a unidade recebida do equipamento.

Esta abordagem permite realizar uma validação automática dos pontos *BACnet*, garantindo não apenas que os valores estão dentro dos limites definidos, mas também que as unidades configuradas nos pontos correspondem às especificações.

```
pv = await bacnet.read(
    f"{target_ip} {pt.tipo} {pt.bac_id} presentValue"
)
units = await bacnet.read(
    f"{target_ip} {pt.tipo} {pt.bac_id} units"
)

pt.present_value = pv
# --- teste de limites ---
if isinstance(pv, (int, float)) and not math.isnan(pv):
    if pv < pt.low_limit or pv > pt.high_limit:
        mensagens.append(
            f"Valor fora dos limites [{pt.low_limit}, {pt.high_limit}]"
        )
    else:
        mensagens.append("presentValue inválido")

# --- teste de unidade ---
if pt.unit and units:
    u_excel = str(pt.unit).strip().lower()
    u_lido = str(units).strip().lower()

    if u_excel != u_lido:
        mensagens.append(
            f"Unidade incorreta (esperado={pt.unit}, lido={units})"
        )
```

Figura 44. Verificação de limites num ponto analógico de entrada.

A figura 45 contém um exemplo de um pequeno teste onde são mostrados os resultados.



Figura 45. Exemplo do resultado de um Teste de Limites.

Quando o teste termina, aparece uma janela que permite guardar um relatório com os resultados do teste.

Dos testes implementados este é o que permite a maior redução de tempo na verificação, pela leitura e aprovação dos pontos ser completamente automática.

4.3.2. Teste nas entradas digitais

O teste da figura 46 pretende avaliar o bom funcionamento das entradas digitais a partir da deteção de mudanças de estado. O teste inclui os pontos do tipo *binary-input* e *binary-value*.

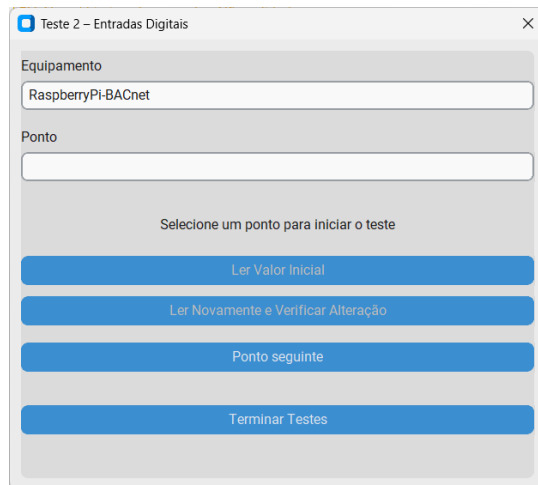


Figura 46. Janela de teste das entradas digitais.

Para que sejam mostrados apenas os pontos do tipo desejado, cada teste tem uma lista de tipos de pontos válidos. O código da figura 47 é responsável por criar dinamicamente a lista de pontos disponíveis para seleção dentro do equipamento atualmente escolhido pelo técnico. Inicialmente, o ciclo percorre todos os pontos pertencentes ao equipamento selecionado (*self.eq_selecionado.pontos*). Em seguida, é realizada uma validação do tipo de objeto. Caso o tipo do ponto não corresponda aos tipos suportados pelo teste em execução, neste caso, *binaryInput* e *binaryValue*, o ponto é ignorado e não é adicionado à lista. Este mecanismo é usado em todos os testes que exigem a seleção de equipamentos e pontos a partir de uma lista na interface gráfica.

```
def _abrir_lista_pontos(self, event=None):
    if not self.eq_selecionado:
        messagebox.showwarning("Aviso", "Selecione primeiro um equipamento.")
        return

    popup = ctk.CTkToplevel(self)
    popup.title("Selecionar Ponto")
    popup.geometry("380x320")
    popup.transient(self)
    popup.grab_set()

    frame = ctk.CTkScrollableFrame(popup)
    frame.pack(fill="both", expand=True, padx=10, pady=10)

    for pt in self.eq_selecionado.pontos:
        if pt.tipo not in self.TIPOS_VALIDOS:
            continue

        label = f"{pt.nome} (ID={pt.bac_id})"
```

Figura 47. Código para a escolha de um ponto da lista de pontos.

Ao selecionar um ponto, a ferramenta monitoriza a entrada selecionada à espera de uma mudança de estado. Quando esta alteração é detetada, o ponto é validado.

O código apresentado na figura 48 implementa a lógica de validação automática de entradas digitais, através da deteção de alterações no estado dos pontos. O processo

inicia-se com um ciclo contínuo de monitorização. Dentro desse ciclo, é criada uma função assíncrona responsável por executar a leitura do valor do ponto selecionado, sem bloquear a interface gráfica. Após a conclusão da leitura, o valor atual do ponto (*pv*) é armazenado na variável *present_value* do ponto selecionado. Caso seja a primeira leitura efetuada, o valor recebido é também guardado como valor inicial (*valor_inicial*), para servir de referência para a validação, pois ela baseia-se na comparação entre o valor atual e o valor inicial do ponto. Sempre que o valor lido seja diferente do estado inicial, considera-se que ocorreu uma mudança de estado válida. Nessa situação, o ponto é automaticamente marcado como aprovado sendo igualmente emitida uma mensagem de confirmação que indica o valor inicial e o valor atual detetado. Após a validação termina o ciclo de leitura.

Caso não seja detetada qualquer alteração de estado, o ponto permanece em monitorização e é emitida uma mensagem a indicar que o valor atual continua inalterado. Nessa condição, o estado do teste é marcado como reprovado (*fail*) até que seja observada uma mudança de estado válida.

```

..
while self._auto_leitura_ativa:
    async def run():
        return await self._ler_bacnet_async()

    future = asyncio.run_coroutine_threadsafe(run(), self.loop)
    try:
        pv = future.result()
        pt = self.pt_selecionado
        pt.present_value = pv

        if pt.valor_inicial is None:
            pt.valor_inicial = pv

        # verifica alteração
        if pv != pt.valor_inicial:
            pt.estado_teste = "ok"
            msg = f"Alteração detectada (inicial={pt.valor_inicial}, atual={pv})"
            pt.mensagem_teste = msg
            self._parar_leitura_automatica()
        else:
            msg = " "
            if pt.estado_teste != "ok" or pt.estado_teste != "fail":
                pt.estado_teste = "fail"
                msg = f"Sem alteração (valor = {pv})"
                pt.mensagem_teste = msg

    self.after(0, lambda m=msg: self.lbl_info.configure(text=m))

```

Figura 48. Código para leitura e aprovação de entradas digitais.

Ao terminar o teste, aparece uma janela que permite guardar o relatório do teste efetuado. Posteriormente, na interface principal dos testes são mostrados os resultados, à semelhança do teste anterior. A figura 49 contém um pequeno teste realizado.

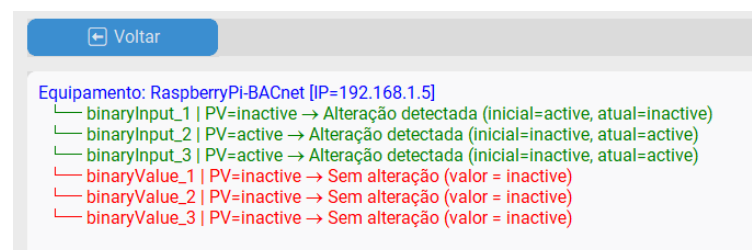


Figura 49. Exemplo de resultado de um teste de entradas digitais.

4.3.3. Teste nas saídas digitais

Este teste pode ser realizado nos pontos do tipo *binary-output* e *binary-value* com o objetivo principal de testar as saídas digitais.

Ao iniciar o teste, é mostrado um aviso de segurança, presente na figura 50, para alertar o técnico que as alterações efetuadas nos pontos são refletidas nas saídas dos equipamentos.

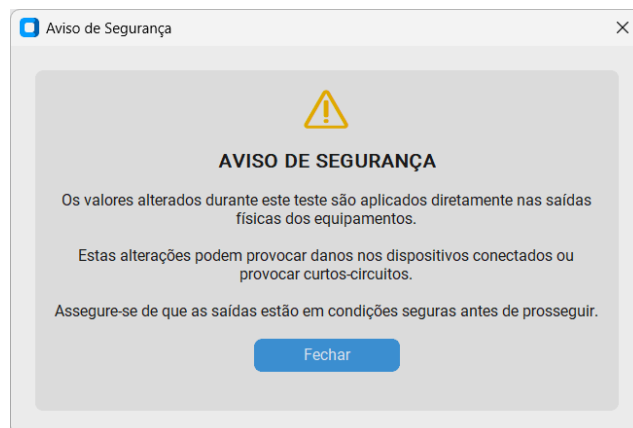


Figura 50. Aviso de segurança dos testes das saídas.

Posteriormente, na interface do teste da figura 51, para realizar uma escrita, deve ser escolhida uma prioridade superior (com menor número) à prioridade que está atualmente a ser usada para escrever no ponto. O teste permite atuar num ponto, ao escrever o valor *ON* ou *OFF*.

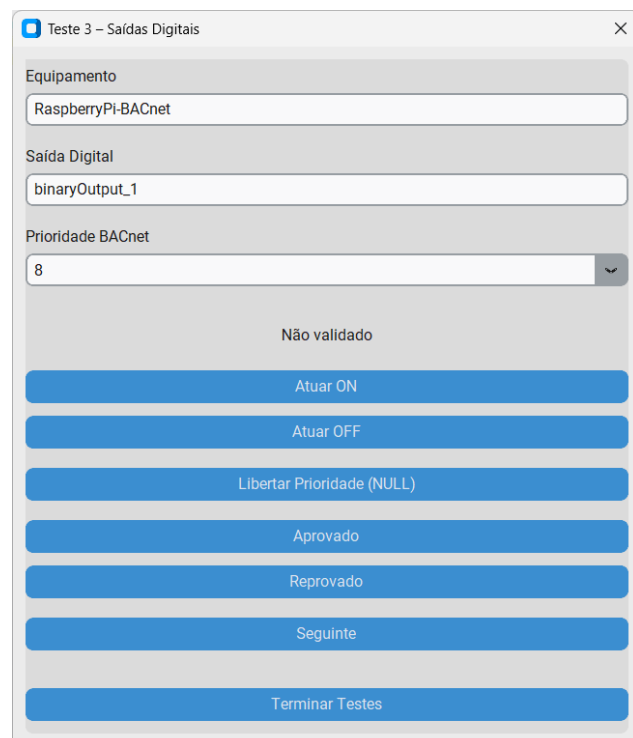


Figura 51. Janela de testes das saídas digitais.

A atuação das saídas digitais é realizada, essencialmente, pelas implementações da figura 52, através de dois botões presentes na interface, correspondentes aos estados

ON e *OFF*. Ao selecionar uma destas opções, é executada a função responsável pela escrita do valor pretendido no ponto *BACnet* selecionado. O valor enviado corresponde aos estados *active* e *inactive*, usados para representar os estados ligado e desligado das saídas digitais. Após selecionar o estado pretendido, a ferramenta obtém a prioridade definida pelo técnico e executa a escrita. Esta função envia o comando para o equipamento através do endereço *IP* do equipamento, o tipo de objeto *BACnet*, o identificador do ponto e a prioridade selecionada.

a)

```

self.btn_on = ctk.CTkButton(
    frame, text="Atuar ON",
    command=lambda: self._escrever_valor("active"),
    state="disabled"
)
self.btn_on.pack(fill="x", pady=5)

self.btn_off = ctk.CTkButton(
    frame, text="Atuar OFF",
    command=lambda: self._escrever_valor("inactive"),
    state="disabled"
)
self.btn_off.pack(fill="x", pady=5)

```

b)

```

def _escrever_valor(self, valor):
    async def run():
        try:
            prioridade = int(self.cmb_prio.get())
            pt = self.pt_selecionado

            pv = await write_with_oos(
                self.bacnet,
                target_ip=self.eq_selecionado.ip,
                obj_type=pt.tipo,
                obj_id=pt.bac_id,
                value=valor,
                priority=prioridade
            )

```

Figura 52. Escrita de valores nos pontos digitais: a) configuração da interface gráfica b) função para escrita.

A validação deve ser feita pelo técnico ao carregar nos botões “Aprovado” ou “Reprovado”, pois normalmente é necessária uma inspeção visual na saída para verificar o resultado da atuação. O código da figura 53 mostra como é feita a validação. Esta abordagem é semelhante aos testes anteriores em que são usadas as variáveis presentes na classe Ponto para guardar os resultados dos testes e respetivas mensagem para, posteriormente, criar os relatórios.

```

def _validar(self, estado):
    pt = self.pt_selecionado
    pt.estado_teste = estado
    pt.mensagem_teste = (
        "Validado pelo técnico."
        if estado == "ok"
        else "Reprovado pelo técnico."
    )
    self.lbl_info.configure(text=pt.mensagem_teste)

```

Figura 53. Código para validação de um ponto.

Já foi mostrado anteriormente como é feita a realização de uma leitura ou escrita, além disto, importa explicar o mecanismo de limpeza de prioridades. Durante o teste, quando há uma troca de ponto, ocorre uma limpeza da última prioridade em que houve uma escrita. Contudo, após a conclusão dos testes é executado um processo de limpeza das prioridades usadas durante os testes. Esta etapa é fundamental para garantir que os equipamentos regressam ao seu funcionamento normal, de modo a evitar que valores escritos temporariamente permaneçam ativos na tabela de prioridades dos objetos *BACnet*. Para tal, o programa efetua a limpeza conforme descrito no fluxograma da figura 54 em que percorre todos os equipamentos e respetivos pontos existentes. São apenas considerados os pontos que participaram efetivamente nos testes. Cada ponto tem uma lista de prioridades que armazena todas

as prioridades onde foram efetuadas escritas ao longo da execução dos testes, o que permite garantir que todas as alterações realizadas são posteriormente removidas. Depois de obtida a lista completa de prioridades, a função percorre cada prioridade individualmente e executa uma escrita com o valor "null" para remover o valor presente nessa posição da tabela de prioridades. Este mecanismo garante que todos os pontos usados durante os testes regressem ao seu estado original após a conclusão dos testes, reduzindo o risco de interferência no funcionamento normal da instalação.

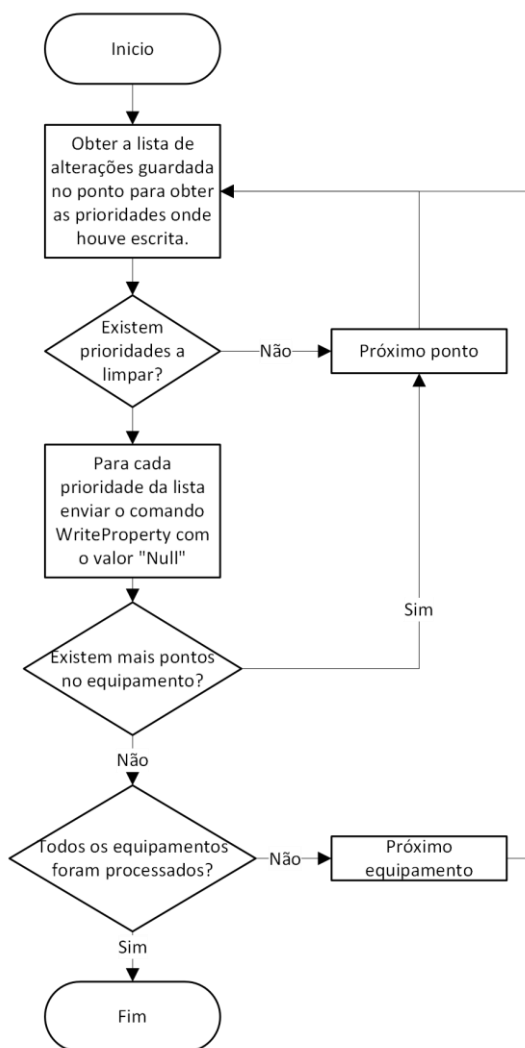


Figura 54. Fluxograma da limpeza de prioridades.

O botão de libertar prioridade permite que seja removido um valor escrito anteriormente na prioridade selecionada.

Ao terminar os testes, é mostrada uma janela que permite guardar os resultados do teste e, posteriormente, na interface principal são mostrados os mesmos resultados à semelhança dos testes anteriores.

4.3.4. Testes nas saídas analógicas

Este teste é semelhante ao anterior e pode ser aplicado a pontos do tipo *analog-output* e *analog-value*, sendo que a diferença reside no tipo de valor a ser escrito. Inicialmente aparece o mesmo alerta e no fim a janela para guardar o relatório dos testes efetuados. Também ocorre a limpeza de prioridades como descrito no teste anterior.

A interface gráfica é semelhante à anterior, sendo que, em vez dos botões para a troca de estado, existe uma caixa de texto para que seja escrito o valor numérico a alterar.

Antes de efetuar a escrita, o código da figura 55 realiza uma validação do valor introduzido pelo técnico, para garantir que apenas valores numéricos sejam escritos no ponto do equipamento. O algoritmo tenta converter o conteúdo introduzido na interface gráfica para um valor numérico do tipo *float*. Caso o utilizador introduza um valor inválido, como texto ou caracteres não numéricos, é gerada automaticamente uma exceção do tipo *ValueError* em que a aplicação apresenta uma mensagem de aviso ao utilizador através de uma janela a indicar que deve ser introduzido um valor numérico válido.

```
def _escrever_valor(self):
    async def run():
        try:
            valor = float(self.entry_valor.get())
            prioridade = int(self.cmb_prio.get())
            pt = self.pt_selecionado

            await write_with_oos(...)

            pt.present_value = valor
            pt.prioridade = prioridade
            # evitar duplicados
            if prioridade not in pt.prioridades_escritas: ...

            self.after(0, lambda: self.lbl_info.configure(...))

        except ValueError:
            self.after(0, lambda: messagebox.showwarning(
                "Valor inválido", "Introduza um valor numérico."
            ))
```

Figura 55. Código para garantir a escrita de um valor numérico.

A validação de cada ponto é realizada manualmente pelo técnico através da seleção dos botões “Aprovado” ou “Reprovado” na interface gráfica. Esta intervenção é necessária porque a confirmação do resultado não pode ser baseada apenas na informação disponibilizada pelo sistema, exigindo uma verificação direta do comportamento da saída em campo. O técnico regista o resultado após confirmar o efeito da atuação no equipamento associado ao ponto.

Após a conclusão dos testes, todas as prioridades que foram usadas para impor valores nos objetos são libertadas através da escrita de valor “*null*” na respetiva prioridade, como foi visto nos testes das saídas digitais.

No final dos testes, é apresentada uma janela dedicada à exportação dos resultados, através de um relatório. Após a gravação, os resultados ficam igualmente disponíveis na interface principal de testes.

4.3.5. Teste de pontos *multistate*

Os pontos *multistate* são objetos usados para representar equipamentos ou processos que podem assumir vários estados. Ao contrário dos pontos binários, que apenas permitem dois estados possíveis (ligado ou desligado), os pontos *multistate* usam números inteiros para representar diferentes condições de funcionamento. Cada valor inteiro corresponde a um estado específico previamente definido no equipamento ou na programação do *BMS*. Por exemplo, um ventilador pode utilizar o valor 1 para “Parado”, 2 para “Baixa velocidade”, 3 para “Velocidade média” e 4 para “Alta velocidade”. Este tipo de ponto é frequentemente usado em *BMS* para representar modos de operação, velocidades ou estados de alarmes.

Este teste permite alterar o estado ao realizar a escrita numérica que corresponde a um determinado modo de funcionamento. É semelhante ao teste das saídas analógicas, sendo que apenas muda o tipo de escrita realizada, pois esta escrita apenas suporta valores inteiros. Já o teste anterior suporta valores decimais. Primeiro ocorre a conversão para o tipo numérico *int* para garantir que o valor corresponde efetivamente a um número inteiro. Também é verificado se o valor introduzido é superior a zero, uma vez que os estados são números inteiros positivos. Caso o técnico introduza um valor inválido, como texto, caracteres especiais, números decimais ou valores inferiores ou iguais a zero, é criada automaticamente uma exceção que impede a continuação da operação de escrita. Esta abordagem evita o envio de dados incorretos para os equipamentos e reduz a probabilidade de erros de funcionamento ou de comportamentos inesperados durante os testes.

Após testar cada ponto, o técnico faz uma inspeção visual do resultado obtido e, com base nessa observação, classifica o ponto como “Aprovado” ou “Reprovado” na interface da gráfica.

Tal como nos restantes testes que envolvem operações de escrita, no final dos testes aos pontos *multistate* é igualmente executado um processo automático de limpeza das prioridades que foram usadas, através da função presente na figura 54. Este procedimento garante que valores escritos temporariamente durante os testes sejam removidos da tabela de prioridades do objeto, para que o equipamento volte ao seu funcionamento normal após a conclusão das validações. Desta forma, evita-se que estados de teste permaneçam ativos no sistema.

Concluídos os testes, é mostrada uma janela onde é possível proceder ao armazenamento do relatório com os resultados dos testes. Depois de guardados, os resultados são enviados para a interface principal, mantendo o formato de visualização usado nos testes anteriores.

4.3.6. Testes Sequenciais

Até o momento, os testes implementados visavam a atuação ou leitura individual de cada ponto. Este teste sequencial foi criado com o objetivo de atuar num conjunto de pontos e verificar o comportamento de outro conjunto de pontos face a esta alteração no estado do sistema.

Na figura 37 é possível visualizar um exemplo de preenchimento da configuração de testes. Um Teste Sequencial é definido por um identificador numérico, a ação que o programa deve aplicar ao ponto (escrita - *Write* ou leitura - *Read*) e o resultado esperado para os pontos lidos através dos sinais “+” e “-“. Os sinais de “+” e de “-“

representam respetivamente um aumento/ativação e uma diminuição/inativação do valor lido no ponto. Ou seja, se for um ponto de leitura analógico, o sinal + indica que se espera, após atuar nos pontos de escrita, um aumento do valor lido no ponto. Caso seja um ponto de leitura digital, o sinal de + indica que, após atuar nos pontos de escrita, é esperado a leitura *active* no ponto. O inverso ocorre para o sinal negativo.

A figura 56 contém o cabeçalho da janela dos testes. Ao selecionar o identificador de um teste é possível escrever ou alterar uma descrição que pode ser guardada para utilização futura. As descrições são guardadas num ficheiro .csv na pasta *assets* do programa. Isto permite que, ao voltar a abrir o programa, não seja necessário reescrever a descrição de cada teste. As descrições servem para indicar o objetivo ou assunto do teste executado. O campo Prioridade permite escolher uma prioridade para escrever o valor em cada ponto. Esta prioridade deve ser numericamente inferior (mais prioritária) àquela que é usada para a escrever no ponto normalmente. Após terminar o teste, o valor escrito na prioridade é removido para que o fluxo de funcionamento do sistema volte ao normal através da prioridade seguinte com valor definido no *array* de prioridades.

Figura 56. Cabeçalho da janela de Testes Sequenciais.

Na interface é possível selecionar dois tipos de testes. O primeiro tipo é a Validação, que permite atuar nos pontos que foram definidos como *Write* e verificar o resultado obtido nos pontos definidos como *Read* após um intervalo de espera definido na interface (*Delay* da figura 56). O segundo tipo de teste é o Limites, que após atuar nos pontos definidos como *Write*, regista o valor inicial e o máximo e mínimo nos pontos definidos como *Read* até que o técnico termine o teste. Posteriormente é mostrada uma janela com os resultados obtidos, sendo que cabe ao técnico aprovar ou reprovado cada ponto individualmente, como ilustra a janela da figura 57.

Ponto	Valor Inicial	Valor Mínimo	Valor Máximo	Status
analogInput_1 analogInput 100	21.618860244750977	21.618860244750977	22.327524185180664	Aprovado
analogInput_2 analogInput 101	13.456472396850586	13.456472396850586	14.122618675231934	Reprovado
analogInput_3 analogInput 102	20.696542739868164	20.696542739868164	21.538206100463867	Aprovado
analogInput_4 analogInput 103	28.25787925720215	28.25787925720215	28.71205711364746	Aprovado
analogInput_5 analogInput 104	17.359548568725586	16.990468978881836	17.63233184814453	Aprovado
binaryInput_1 binaryInput 105	0.0	0.0	1.0	Reprovado
binaryInput_2 binaryInput 106	0.0	0.0	0.0	Aprovado
binaryInput_3 binaryInput 107	0.0	0.0	0.0	Aprovado

Figura 57. Exemplo de validação manual de um Teste Sequencial do tipo Limites.

A implementação dos dois testes segue uma estrutura comum até à fase de atuação sobre os pontos configurados como *Write*, onde é realizada a escrita dos valores definidos pelo utilizador nos respetivos objetos. Nesta etapa inicial, são percorridos todos os pontos associados ao teste e, para cada um deles, é executada a operação de escrita na prioridade definida na interface gráfica. O código para esta primeira fase encontra-se na figura 58. Para cada ponto, ocorre a leitura do valor escrito na interface para que seja escrito no ponto, se houver valor. Caso o valor seja inválido, ocorre como nas escritas dos testes anteriores, em que é mostrada uma janela a indicar o erro.

```
# WRITE
for eq, pt, teste, entry, _ in self.write_entries:
    valor = entry.get().strip()

    if valor == "":
        continue

    await write_with_oos(
        self.bacnet,
        target_ip=eq.ip,
        obj_type=pt.tipo,
        obj_id=pt.bac_id,
        value=valor,
        priority=prioridade
    )
    #Guardar resultados para o relatório pdf
    teste.tp_mensagem = f"Escrito: {valor} | Prioridade: {prioridade}"
```

Figura 58. Código de escrita no teste sequencial.

Após esta fase de escrita, o algoritmo segue conforme o tipo de teste selecionado. No caso do teste de Validação, é introduzido um intervalo de espera configurável, permitindo que o sistema estabilize antes da leitura dos pontos definidos como *Read*. Após este tempo, é realizada a leitura do valor atual de cada ponto, conforme o código da figura 59, e, depois, a comparação entre o valor obtido e o resultado esperado que foi definido anteriormente no teste. A validação é dependente do tipo de objeto, ou seja, entre pontos digitais e analógicos, ao aplicar regras específicas para cada caso.

```
for eq, pt, teste, lbl in self.read_labels:
    pt.estado_teste = "pendente"
    pv = await self.bacnet.read(
        f"{eq.ip} {pt.tipo} {pt.bac_id} presentValue"
    )

    #ok = str(pv) == str(teste.tp_resultado)
    # Inicializa a variável
    ok = False
    esperado = "indefinido"
    # Definir tipos digitais conhecidos (ajusta conforme necessário)
    TIPOS_DIGITAIS = ("binaryinput", "binaryoutput", "binaryvalue")
    if pt.tipo.lower() in TIPOS_DIGITAIS:
        # Ponto digital
        pv_str = str(pv).lower()
        if teste.tp_resultado == "+":
            ok = pv_str in ("active", "on", "true", "1")
            esperado = "active"
        elif teste.tp_resultado == "-":
            ok = pv_str in ("inactive", "off", "false", "0")
            esperado = "inactive"
    else:
        # Ponto analógico
        try:
            pv_float = float(pv)
            inicial = float(pt.valor_inicial)
            if teste.tp_resultado == "+":
                ok = pv_float > inicial + 1 #Evita validar oscilações
                esperado = "subida"
            elif teste.tp_resultado == "-":
                ok = pv_float < inicial - 1 #Evita validar oscilações
                esperado = "descida"
```

Figura 59. Implementação para leitura e verificação de resultados do teste de Validação.

No teste de Limites, após a escrita inicial, o sistema entra num ciclo contínuo de leitura dos pontos *Read*, no qual são monitorizados em tempo real os valores desses pontos. Durante este ciclo, são atualizados dinamicamente os valores mínimos e máximos observados para cada ponto, bem como o valor inicial registado no início do teste. Este processo mantém-se ativo enquanto o teste não for interrompido pelo técnico através de um botão de pausa, permitindo assim a recolha contínua do comportamento do sistema ao longo do tempo.

```
while self._limites_running:
    for eq, pt, teste, lbl in self.read_labels:
        try:
            pv = await self.bacnet.read(
                f"{eq.ip} {pt.tipo} {pt.bac_id} presentValue"
            )
            pv_float = float(pv)

            data = limites.get(pt)
            if data["min"] is None:
                continue

            if pv_float < data["min"]:
                data["min"] = pv_float
            if pv_float > data["max"]:
                data["max"] = pv_float

            lbl.configure(
                text=(
                    f"→ Inicial: {data['inicial']} | "
                    f"PV: {pv_float} | "
                    f"Min: {data['min']} | "
                    f"Max: {data['max']}"
                )
            )
            pt.limites_min = data["min"]
            pt.limites_max = data["max"]
            pt.limites_inicial = data["inicial"]
        except Exception:
            lbl.configure(text="Erro")
```

Figura 60. Implementação para o Teste de Limites.

Por fim, à semelhança dos testes anteriores, ocorre a limpeza do *array* de prioridades nas posições em que houve alterações.

A interface dos Testes Sequenciais conta ainda com a capacidade de adicionar e definir novos testes, caso não tenham sido definidos na folha de cálculo. É possível fazer uma procura pelo nome do ponto ou tipo de ponto, seleccionar os pontos que se pretendem adicionar através de *checkbox* e seleccionar se é um ponto em que se pretende atuar ou ler valores. Cada ponto será adicionado ao teste criado apenas se já não tiver um teste com o mesmo identificador.

4.3.7. Monitorização

O último botão presente na parte inferior da interface de testes permite abrir uma janela, ilustrada na figura 61, para registar valores de forma gráfica.

É possível escolher um período de amostragem em segundos e a duração da monitorização em minutos. Todos os pontos importados através do ficheiro *Excel* são monitorizados. Esta interface permite seleccionar um ponto para visualizar os últimos valores amostrados, sendo que todos os outros pontos continuam a ter amostras guardadas.

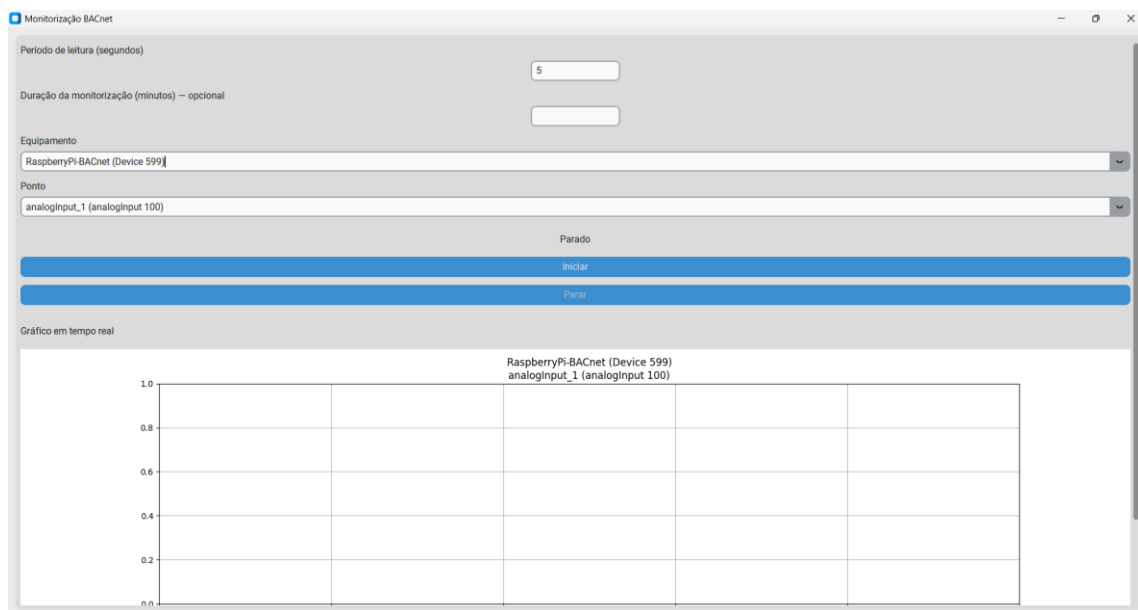


Figura 61. Janela de monitorização.

Ao iniciar a monitorização, escolhe-se o diretório para guardar o ficheiro .csv com todas as amostras recolhidas. A monitorização é executada através de um ciclo contínuo assíncrono, presente na figura 62, que permanece ativo enquanto o registo estiver em execução. Em cada iteração é obtido o instante temporal atual e são percorridos todos os equipamentos válidos. Para cada equipamento, são igualmente percorridos todos os pontos configurados, sendo realizada a leitura da propriedade *presentValue*, ou seja, o valor atual. Os valores obtidos são imediatamente registados juntamente com a informação temporal associada. Caso ocorra uma falha durante a leitura de algum ponto, o erro é capturado e registado sem interromper o restante processo de aquisição de dados. Entre cada ciclo de leitura é aplicado um atraso correspondente ao período de amostragem definido pelo técnico. O ciclo mantém-se ativo até que a monitorização seja interrompida manualmente ou até ser atingido o tempo máximo definido para a monitorização.

```

async def ciclo():
    while self.ativo and self.viva:

        ts = datetime.now().isoformat()

        for eq in self.equipamentos.values():
            if not eq.ip or eq.status == "erro":
                continue

            for pt in eq.pontos:
                if pt.bac_id is None or not pt.tipo:
                    continue

                try:
                    valor = await self.bacnet.read(
                        f"{eq.ip} {pt.tipo} {pt.bac_id} presentValue"
                    )
                except Exception as e:
                    valor = f"ERRO: {e}"

                self._registar(ts, eq, pt, valor)

```

Figura 62. Implementação para registo contínuo de valores.

O mecanismo responsável pela gestão do tempo restante da monitorização encontra-se na figura 63. Quando o técnico define uma duração máxima para a monitorização, o sistema calcula automaticamente o instante previsto para terminar a aquisição de

dados com base na hora de início e soma do tempo definido pelo técnico. A partir desse instante, é efetuado um cálculo contínuo da diferença entre o tempo atual e o tempo final definido, permitindo determinar o tempo restante disponível para a monitorização. Esta informação é atualizada periodicamente e apresentada diretamente na interface gráfica no formato “horas:minutos:segundos”, para permitir ao técnico acompanhar o tempo restante. Quando o tempo restante atinge o valor zero, o sistema termina automaticamente a monitorização e interrompe o ciclo de leituras.

```
def _tick_tempo(self):
    if not self.ativo or not self.instante_fim:
        return

    restante = self._tempo_restante()

    if restante == "00:00:00":
        self._parar()
        return

    self.lbl_estado.configure(
        text=f"A monitorizar... [{restante} restante]"
    )

    self.after(1000, self._tick_tempo)

def _tempo_restante(self):
    total = int((self.instante_fim - datetime.now()).total_seconds())
    if total <= 0:
        return "00:00:00"

    h = total // 3600
    m = (total % 3600) // 60
    s = total % 60

    return f"{h:02d}:{m:02d}:{s:02d}"
```

Figura 63. Funções para controlar o tempo de monitorização.

A figura 64 contém a função usada para guardar cada valor lido durante o ciclo da figura 62. Cada amostra recolhida é registada num ficheiro no formato CSV. Em cada registo são armazenadas informações relativas ao instante da leitura, o nome e identificador do equipamento, bem como o nome, tipo e identificador *BACnet* do ponto, juntamente com o respetivo valor lido. A escrita é efetuada de forma incremental, ao adicionar cada nova amostra ao ficheiro existente sem necessidade de manter todos os dados em memória, pois apenas são guardados no programa os últimos 300 valores do ponto que é mostrado no gráfico. Esta abordagem permite suportar monitorizações prolongadas e um elevado número de pontos sem impacto significativo no consumo de recursos da aplicação. Paralelamente ao registo no ficheiro, o sistema verifica se o ponto atualmente monitorizado na interface gráfica corresponde ao ponto selecionado pelo utilizador para visualização em tempo real. Quando essa condição é satisfeita, e o valor lido é válido, o gráfico é atualizado automaticamente com a nova amostra recolhida. Desta forma, é possível acompanhar visualmente a evolução temporal de um ponto específico enquanto os restantes continuam igualmente a ser registados em segundo plano.

```
def _registar(self, ts, eq, pt, valor):
    with open(self.csv_path, "a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow(
            [ts, eq.nome, eq.bac_id,
             pt.nome, pt.tipo, pt.bac_id, valor]
        )

    if (
        self.viva and
        self.eq_sel == eq and
        self.pt_sel == pt and
        not isinstance(valor, str)
    ):
        self.after(0, lambda v=valor: self._atualizar_grafico(v))
```

Figura 64. Função para armazenar valores lidos durante a monitorização.

Na figura 65 é possível observar o formato usado para guardar as leituras de cada ponto no ficheiro.

```
1 timestamp,equipamento,ip,ponto,tipo,bac_id,valor
2 2026-02-09T16:14:29.692153,RP-C-16A_TestesBACnet,192.168.1.241,Resistive Input,analogInput,18,655.3499755859375
3 2026-02-09T16:14:29.692153,RP-C-16A_TestesBACnet,192.168.1.241,Analog Value,analogValue,52,50.0
4 2026-02-09T16:14:29.692153,ES_0,192.168.1.248,ACC55,analogValue,55,7.0
5 2026-02-09T16:14:29.692153,ES_0,192.168.1.248,ACC56,analogValue,56,0.0
6 2026-02-09T16:14:29.692153,ES_JN,192.168.1.219,C01_EnT,analogValue,9101,50.0
7 2026-02-09T16:14:29.692153,ES_JN,192.168.1.219,C02_EnT,analogValue,9102,25.0
8 2026-02-09T16:14:29.692153,ES_JN,192.168.1.219,C03_EnT,analogValue,9103,36.0
9 2026-02-09T16:14:29.692153,ES_JN,192.168.1.219,C04_EnT,analogValue,9104,10.0
```

Figura 65. Exemplo de ficheiro de monitorização.

Para visualizar os resultados da monitorização após a gravação dos dados, é preciso usar o programa “Graficos.py”. Neste programa é possível abrir o ficheiro .csv e escolher os equipamentos e pontos para visualizar os gráficos. A figura 66 contém o resultado de um teste de continuidade realizado durante oito horas que serviu para testar a estabilidade da conexão da ferramenta. Foram registados dados de sete pontos *BACnet* de um equipamento com uma amostra por minuto por ponto.

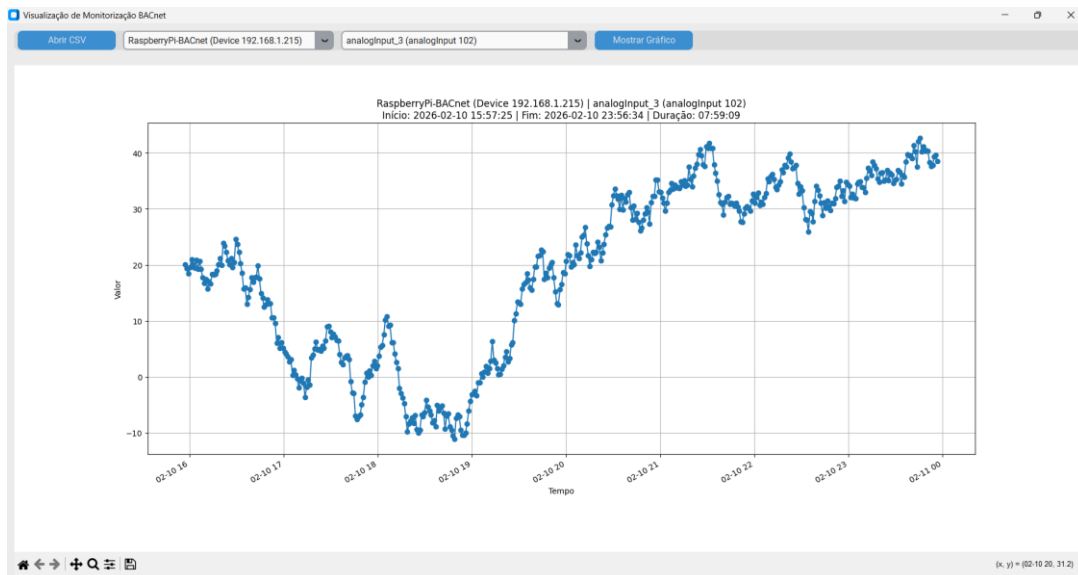


Figura 66. Gráfico de amostragem de um ponto *BACnet*.

O desenvolvimento da aplicação mostrada na figura anterior tem como base os conceitos da ferramenta principal: interface gráfica, listas de equipamentos e pontos, representação gráfica.

4.4. Criação automática de relatórios para os testes

Ao fim de cada teste, a ferramenta disponibiliza ao técnico a possibilidade de guardar um relatório com base nos dados armazenados. Todos estes relatórios têm um modelo semelhante. No topo do relatório o nome da instalação, técnico responsável e data da realização do teste. Posteriormente, como ilustra a figura 67, há um resumo do teste com a indicação do número de pontos e equipamentos com alertas ou erros.

Resumo da Validação

	OK	Alterado	Erro
Equipamentos	3	0	2
Pontos	91	1	24

Figura 67. Exemplo de resumo de um teste de presença.

Por fim, seguem-se as tabelas com os detalhes do teste, à semelhança da figura 68.

Equipamento: RP-C-16A_TesteBACnet (IP=192.168.1.241, BACnetID=100527) [OK]

Ponto	Tipo	BACnet ID	Estado	Mensagens
Resistive Input	analogInput	18	ok	-
Analog Value	analogValue	52	ok	-
Digital Input	binaryInput	31	alterado	BACnet ID atualizado pela rede para 31)
c	-	None	erro	Ponto não encontrado no equipamento
Digital Output	binaryOutput	35	ok	-
Digital Value_2	binaryValue	36	ok	-

Figura 68. Exemplo de detalhes do teste de presença num equipamento.

Para além disso, os cabeçalhos e rodapés contam com informações relativas ao nome da empresa, instalação, *timestamp*, e numeração de páginas. Já o nome com o qual o ficheiro é guardado permite a identificação do tipo de teste, instalação e versionamento através do *timestamp*:

Relatorio_{tipo_de teste}_{instalação}_{timestamp}.pdf

Para criar os relatórios foi usada a biblioteca *reportlib* que já contém todas as ferramentas necessárias para criar diferentes estruturas como as tabelas.

Para a construção da tabela de resumo da figura 67 é usada a função da figura 69. A função percorre todos os equipamentos para verificar a informação relativa ao estado do teste guardada na classe do equipamento. De forma análoga, é igualmente realizada a verificação ao nível dos pontos individuais. O processo descrito baseia-se exclusivamente na leitura dos estados já atribuídos durante a execução dos testes, não sendo necessário qualquer processamento adicional dos dados.

```
def calcular_resumo(equipamentos):
    eq = {"ok": 0, "alterado": 0, "erro": 0}
    pt = {"ok": 0, "alterado": 0, "erro": 0}

    for e in equipamentos.values():
        eq[e.status] += 1
        for p in e.pontos:
            pt[p.status] += 1

    return eq, pt
```

Figura 69. Função para cálculo do resumo de um relatório.

A tabela de resumo dos pontos de um equipamento da figura 63 é criada a partir da lista de pontos associados ao equipamento. Primeiro é definida uma linha inicial com os títulos das colunas, onde são apresentados o nome do ponto, o tipo, o identificador *BACnet*, o estado e as mensagens associadas ao teste. De seguida, o código da figura 70 percorre todos os pontos do equipamento. Para cada ponto, é obtida a informação necessária para preencher a linha da tabela. O estado do ponto é usado para definir a cor apresentada na tabela, para identificar rapidamente se o ponto passou, falhou ou está em outro estado. Caso existam várias mensagens de teste, são unidas numa única célula com quebras de linha para facilitar a leitura.

```
tabela = [{"Ponto", "Tipo", "BACnet ID", "Estado", "Mensagens"}]

for pt in eq.pontos:
    cor_pt = CORES_ESTADO.get(pt.status, colors.black)
    if pt.mensagens:
        msgs = Paragraph(
            "<br/>".join(pt.mensagens),
            styles["Texto"]
        )
    else:
        msgs = Paragraph("-", styles["Texto"])

    tabela.append([
        Paragraph(pt.nome or "-", styles["Texto"]),
        Paragraph(pt.tipo or "-", styles["Texto"]),
        Paragraph(str(pt.bac_id), styles["Texto"]),
        Paragraph(
            f"<font color='{cor_pt.hexval()}'>{pt.status}</font>",
            styles["Texto"]
        ),
        msgs
    ])
```

Figura 70. Código para criar a tabela com informação dos testes dos pontos de cada equipamento.

4.5. Adaptações para o ambiente *Linux*

Durante a fase final do desenvolvimento, a ferramenta foi adaptada para execução num sistema Linux, especificamente no *Raspberry Pi OS*, para facilitar a sua implementação de forma permanente numa instalação através de um equipamento leve, portátil e de baixo consumo. Este equipamento tem a capacidade de ser instalado num quadro de automação com recurso a uma caixa como a da figura 71 com suporte a encaixe para calha *DIN*. Esta adaptação implicou algumas alterações técnicas devido às diferenças entre os ambientes *Windows* e *Linux*, principalmente na interface gráfica.



Figura 71. Caixa de calha DIN para *Raspberry Pi* [41].

Inicialmente, a aplicação foi desenvolvida e testada em ambiente Windows, com *Python 3.11* e diversas bibliotecas externas, incluindo as bibliotecas *BAC0*, *pandas*, *matplotlib*, *customtkinter* e *xlwings*. No Windows, a biblioteca *xlwings* comunicava diretamente com o *Microsoft Excel* através da interface *COM* (*Component Object Model*). No entanto, esta interface não estava disponível nos sistemas Linux, o que impossibilitou a utilização direta do *Excel* para manipulação de ficheiros *.xlsx*. A funcionalidade originalmente baseada na biblioteca *xlwings* foi substituída por uma abordagem baseada nas bibliotecas *pandas* e *openpyxl*, permitindo a leitura e escrita direta de ficheiros *.xlsx* sem depender de aplicações externas. Esta solução apresentava a vantagem de ser totalmente multiplataforma e não requerer a instalação de *software* adicional como *Microsoft Excel*. Contudo, perdeu-se uma funcionalidade que permitia guardar o ficheiro *Excel* com os dados atualizados pela ferramenta com base na informação da rede. Ainda era possível guardar os dados atualizados, contudo, a formatação original era perdida.

Outra adaptação necessária está relacionada com a interface gráfica. A aplicação utiliza a biblioteca *CustomTkinter* para a criação das interfaces. Algumas funcionalidades disponíveis no *Windows* não estavam implementadas da mesma forma no *Linux*. Por exemplo, o estado de janela *zoomed* usado para abrir janelas maximizadas no *Windows* não era suportado no *Tkinter* para *Linux*. Para resolver esta limitação, a maximização da janela passou a ser realizada através da definição manual da geometria da janela com base na resolução do ecrã.

Adicionalmente, verificaram-se diferenças no comportamento de algumas janelas. No *Linux*, a utilização da função *grab_set()* (bloqueia a janela principal e garante a utilização da janela secundária pelo utilizador) nas janelas secundárias exigia que a janela estivesse previamente visível. Caso contrário, ocorria um erro de execução indicando que a janela ainda não era *viewable*. Para garantir o funcionamento correto, foi necessário forçar a atualização da interface gráfica antes da aplicação da função *grab_set()*, através da função *update()*.

A figura 72 contém um excerto de código relativo à inicialização de uma janela de um dos testes. Nela é possível visualizar que o tamanho da janela passou a ser definido manualmente, em vez de ficar em tela cheia automaticamente e a existência da função *update()* antes da função *grab_set()*.

```

class TesteDigitalIn(ctk.CTkToplevel):
    """
    Teste 2  Entradas Digitais
    """
    TIPOS_VALIDOS = (
        "binaryValue",
        "binaryInput"
    )

    def __init__(self, parent, equipamentos: Dict, bacnet, loop):
        super().__init__(parent)
        self.title("Teste 2  Entradas Digitais")
        self.geometry("500x420")
        self.transient(parent)
        self.update()
        self.grab_set()

```

Figura 72. Adaptação da interface gráfica para ambiente *Linux*.

Por fim, para facilitar a execução da aplicação no *Raspberry Pi*, foi criado um script de inicialização em *Bash* que ativava automaticamente o ambiente virtual *Python* e executa o programa principal.

Estas adaptações permitiram garantir o funcionamento completo da aplicação no *Raspberry Pi OS*, mantendo as funcionalidades desenvolvidas originalmente no ambiente *Windows*, exceto a funcionalidade que permite guardar o ficheiro *Excel* com os dados atualizados pela ferramenta com base na informação da rede, assegurando a portabilidade do *software* entre diferentes sistemas operativos.

4.6. Testes intermédios e finais

Nesta secção serão descritos os testes realizados para atestar o bom e correto funcionamento da ferramenta desenvolvida. O primeiro teste teve como objetivo testar algumas particularidades da ferramenta e verificar o seu funcionamento. Já o segundo teste decorreu durante duas manutenções em dois hotéis com o objetivo de testar a ferramenta em grande escala e verificar quais as melhorias e correções a implementar.

Após a realização de todos os testes, fez-se um manual de instalação e utilização desta ferramenta.

4.6.1. Teste intermédio

Após a implementação das funcionalidades base da ferramenta, pretendeu-se realizar alguns testes para aferir quais as melhorias que teriam de ser feitas. Este teste foi realizado no Hotel *The Views Oásis*, doravante Oásis, no município de Santa Cruz, na freguesia do Caniço.

O quadro de automação do Oásis encontra-se parcialmente representado na figura 73, onde constam 31 equipamentos. Estes equipamentos estavam todos ligados em rede. Inicialmente, usou-se o programa *YABE* para descobrir estes equipamentos na rede e guardar as informações necessárias para o ficheiro *Excel* a ser importado na ferramenta. Não foram importados todos os equipamentos e pontos, mas apenas alguns com características peculiares e de interesse para este teste.



Figura 73. Quadro de automação do Hotel *The Views Oásis*.

Foram analisadas, essencialmente, duas características: a capacidade da ferramenta de identificar equipamentos hierarquicamente ligados a outro equipamento mestre e a capacidade da ferramenta de identificar pontos diferentes com o mesmo identificador numérico *BACnet*. A primeira situação pode ser visualizada no *YABE* da figura 74. Nele é possível observar que o acesso ao *CHILLER1* e ao *CHILLER2* era feito por um *AS-P*.

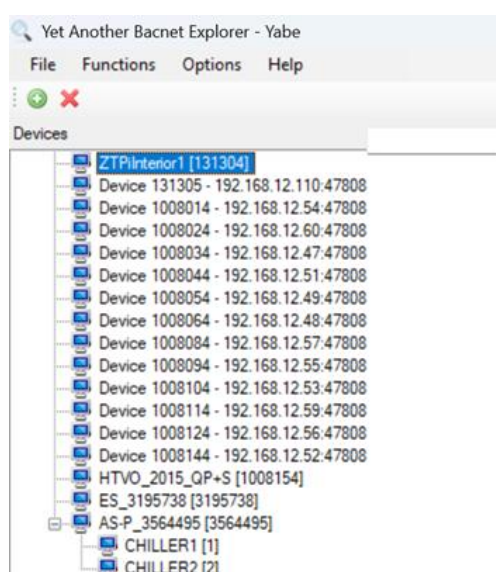


Figura 74. Lista de equipamentos do Hotel *The Views Oásis* no software *YABE*.

A partir da figura 75 é possível observar os resultados que demonstram estas duas situações. Foi possível para a ferramenta identificar os equipamentos subordinados a outro. Contudo, a ferramenta não conseguiu identificar pontos *BACnet* com o mesmo identificador numérico, pela forma como o algoritmo realizava a procura dos pontos na rede. Posteriormente este erro foi corrigido para permitir pontos de diferentes tipos com o mesmo identificador.

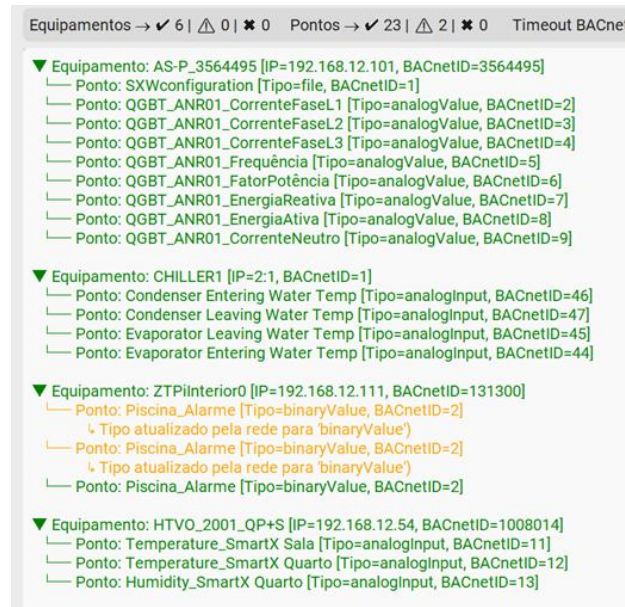


Figura 75. Resultados dos testes no Hotel *The Views Oásis*.

Implementou-se a correção presente no excerto de código da figura 76. O programa verifica se existem vários candidatos com o mesmo identificador. A diferenciação é feita pelo tipo do ponto. Isto obriga a que no ficheiro *Excel* seja identificado o tipo de ponto nos casos em que existam pontos com o mesmo identificador *BACnet*.

```
elif len(candidatos) > 1:
    # Ambiguidade → usar tipo se existir
    if pt.tipo:
        tipo_norm = pt.tipo# normalizar_tipo_bacnet(pt.tipo)

        for obj in candidatos:
            obj_tipo_norm = normalizar_tipo_bacnet(obj["type"])
            if obj_tipo_norm == tipo_norm:
                rede = obj
                encontrado = True
                # Se veio pelo BACnet ID e não tem nome, preencher a partir da rede
                if not pt.nome and "name" in rede:
                    pt.nome = rede["name"]
                break
            else:
                pt.adicionar_mensagem(
                    f"ID {pt.bac_id} é ambíguo na rede. "
                    f"Indique o tipo do ponto no Excel.",
                    status="erro"
                )
        continue
```

Figura 76. Implementação para permitir a identificação de pontos diferentes com o mesmo identificador.

4.6.2. Testes finais e melhorias implementadas

Os testes finais foram realizados no Hotel Alto Lido e no Hotel Baía Azul, pertencentes ao Grupo Cardoso, ambos no município do Funchal, freguesia de São Martinho. A escolha foi devido a uma manutenção que a empresa Blin Control tinha agendada para estes hotéis próxima ao momento em que a ferramenta ficou pronta para testes.

A maior parte dos quadros de automação dos hotéis não usavam pontos *BACnet*. Logo, foi necessário criar estes pontos e fazer a ligação aos pontos físicos dos equipamentos. Para tal, usou-se a ferramenta do capítulo 3 para criar o ficheiro *XML*

com os pontos *BACnet* e a respetiva ligação aos pontos físicos. Após a criação de todos os ficheiros com os pontos *BACnet* necessários, o engenheiro responsável pela manutenção, importou-os para o computador de cada hotel onde estava instalado o *EBO*. A figura 77 contém um excerto onde é possível visualizar os equipamentos de um quadro de automação e alguns dos pontos *BACnet*.



Figura 77. Importação de pontos *BACnet* num quadro de automação.

Terminado o procedimento anterior, procedeu-se à instalação da ferramenta nos computadores referidos anteriormente. O processo consistiu na instalação do *Python*, das bibliotecas nas versões adequadas, através de um *script* preparado para tal, e à transferência dos ficheiros do programa. A instalação teve de ser feita no computador local de cada hotel, tendo em conta as normas de segurança estabelecidas, em que não era permitido ligar computadores externos à rede. A figura 78 contém o esquema que representa a forma como foram feitos os testes. Usou-se uma ligação remota com interface gráfica para aceder ao computador do hotel com o programa de testes instalado. Assim, foi possível aceder aos equipamentos na mesma rede desse computador.

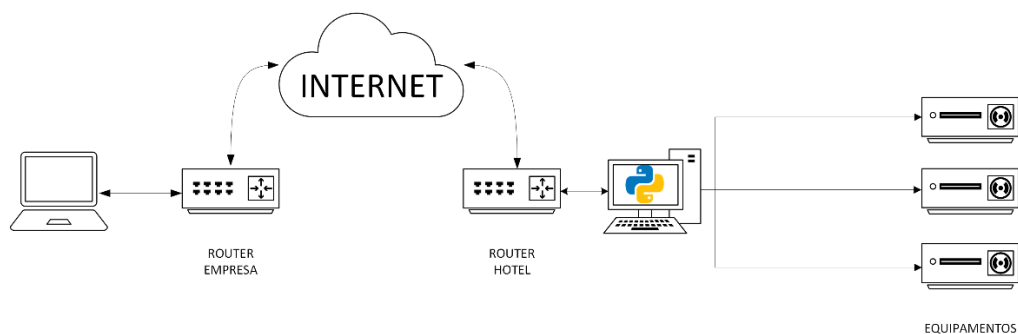


Figura 78. Esquema de ligação remota aos equipamentos dos hotéis.

Foram realizados testes de forma a avaliar todas as funcionalidades da ferramenta desenvolvida. Foram encontrados dois problemas que não estavam relacionados com o bom funcionamento da ferramenta de testes, mas sim com a ferramenta que criou os ficheiros *XML* com os pontos *BACnet* para importação.

O primeiro problema está evidenciado na tabela da figura 79 retirada do relatório do Teste de Limites. Apesar do ponto físico ter a sua unidade configurada no ficheiro XML, o ponto *BACnet* não herda a unidade deste ponto, embora a existência de uma ligação com os pontos físicos, logo, os valores lidos não apresentavam unidade. Para corrigir a situação, também é necessário definir a unidade no ficheiro XML dos pontos *BACnet*.

Ponto	Tipo	BACnet ID	Valor	Limites	Unidade	Resultado
UTA5_Pressao_Dif	analogInput	149	77.900001 5258789	[25.0, 975.0]	pascals	Unidade incorreta (esperado=pascals, lido=no-units)
UTA5_Temp_Retorno	analogInput	150	23.299999 23706054 7	[10.0, 40.0]	degrees-celsius	Unidade incorreta (esperado=degrees-celsius, lido=no-units)

Figura 79. Problema identificado com as unidades nos pontos *BACnet*.

Para corrigir o problema anterior, na função responsável por criar os pontos *BACnet*, adicionou-se o código da figura 80. A variável “UnidadeBacnet” recebe a unidade proveniente do ponto físico correspondente, tendo em conta que os códigos hexadecimais são os mesmos, presentes na tabela 3 da página 39.

```

if UnidadeBacnet:
    pi.set("Unit", UnidadeBacnet)
# Criar PI adicional para Value quando for AssociatedValue para ter uma unidade no Value
if pi_def["Name"] == "AssociatedValue" and UnidadeBacnet:
    ET.SubElement(
        oi,
        "PI",
        Name="Value",
        Unit=UnidadeBacnet
    )

```

Figura 80. Código para adicionar a unidade a um ponto *BACnet*.

Após aplicar a correção anterior, voltou-se a criar novamente alguns dos ficheiros XML para teste. A figura 81 tem um fragmento de um dos ficheiros onde é possível observar a descrição de um ponto *BACnet* com a informação da unidade. A unidade é colocada apenas em pontos analógicos de entrada, saída ou *analog-values* (sem distinção de entrada ou saída).

```

<OI NAME="UTA5_Pressao_Dif" TYPE="bacnet.point.analog.Input" >
  <PI Name="AssociatedValue" Unit="0x230006" >
    <Reference DeltaFilter="0" Object="../../../IO Bus/Slot06: UI-8.AO-V-4/UTA5_Pressao_Dif" Property="Value" Retransmit="0" TransferRate="10" />
  </PI>
  <PI Name="Value" Unit="0x230006" />
  <PI Name="BACnetName" Value="UTA5_Pressao_Dif" />
  <PI Name="ForeignAddress" Value="analog-input, 149" />
</OI>

```

Figura 81. Ficheiro XML com a criação de um ponto *BACnet* com indicação da unidade.

Ao continuar a testar a ferramenta, identificou-se outro problema ao realizar um teste numa saída. Verificou-se que o valor pretendido foi escrito no ponto *BACnet*, mas o resultado desta mudança não foi visível na saída, ou seja, no ponto físico. Posteriormente, identificou-se que o problema estava na configuração dos ficheiros XML nas referências entre os pontos *BACnet* e pontos físicos. Todas as referências eram feitas como consta na figura 81, cujo resultado no *EBO* encontra-se na figura 82. A seta na figura significa que o valor do ponto físico é passado para o ponto *BACnet*. Este comportamento é adequado para as entradas, em que estes valores

são enviados para os pontos *BACnet*. Para as saídas, o comportamento deve ser o inverso em que o ponto *BACnet* deve colocar um valor na saída.

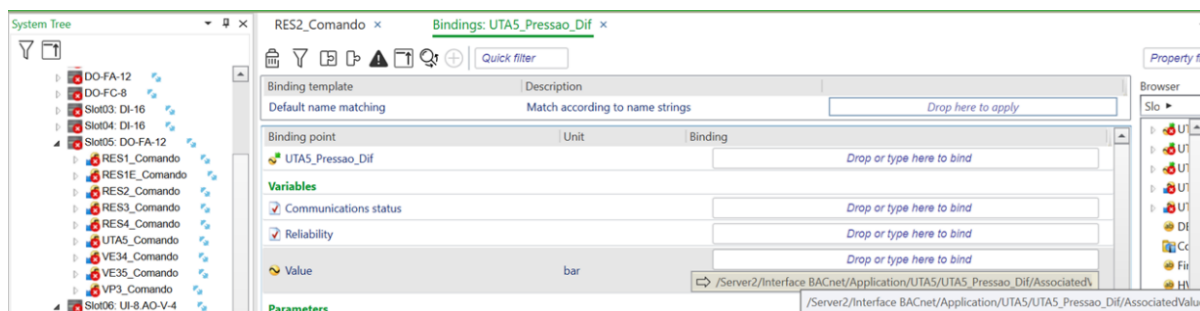


Figura 82. Janela de *bindings* de um ponto físico de entrada.

Contudo, quando é necessário trocar o sentido da referência, ela passa a estar descrita no ficheiro *XML* dos pontos físicos. Logo, as alterações no código devem ter em conta se o ponto físico é uma entrada ou saída, para decidir onde escrever a referência. Até ao momento, todas as referências eram indicadas no ficheiro *XML* dos pontos *BACnet*.

No Anexo I encontra-se a implementação nova e a implementação antiga usadas para escrever as referências. Caso o ponto seja do tipo *analog-output* ou um *analog-value* ligado a uma saída física, a referência é escrita no ficheiro *XML* do ponto físico. No código, “pi” contém a parte da árvore *XML* referente ao *Value* do ponto *BACnet* e “pi_o” a parte da árvore correspondente ao *RequestedValue* do ponto físico. Assim, numa saída, a ligação entre ponto físico e *BACnet* aparece como na figura 83 em que o valor é enviado do ponto *BACnet* para o ponto físico através do campo *RequestedValue*. No campo *Value* pode ser colocada uma referência a um outro ponto *BACnet*, mas que apenas permite fazer a leitura do valor do ponto físico através do ponto *BACnet*.

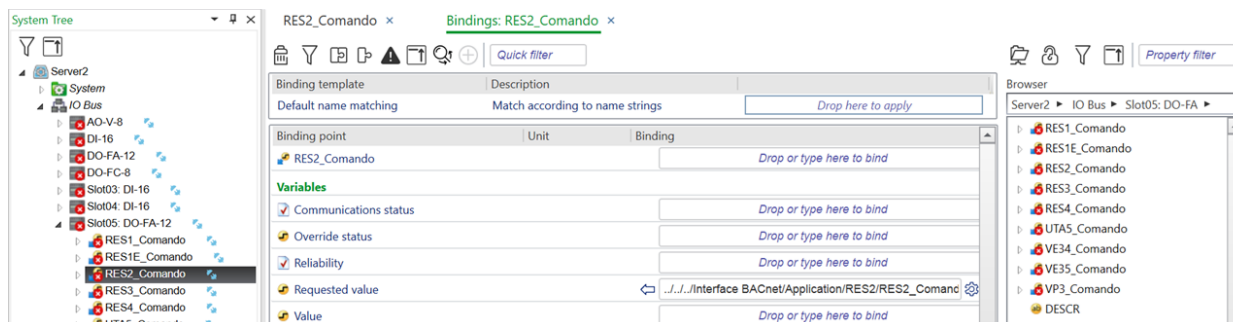


Figura 83. Janela de *bindings* de um ponto físico de saída.

Assim, também é possível construir vários programas que escrevem no ponto *BACnet* com diferentes prioridades, sendo que o mais prioritário é escrito no ponto físico.

Implementadas as correções descritas anteriormente, voltou-se a efetuar alguns testes para verificar o desempenho e bom funcionamento da ferramenta desenvolvida.

Foram realizados alguns testes no Quadro de AVAC 3 (QEAVAC3). Primeiro, fez-se um teste de saídas digitais em 3 pontos de comando que, internamente, eram controlados por programas semelhantes ao da figura 84. Este teste consistiu em alterar o valor destas saídas e verificar a alteração de forma visual. Para conseguir realizar a escrita, na interface da figura 51 da página 62 usou-se a prioridade 8, que

normalmente é a prioridade usada para realizar manutenções. Contudo, caso os programas escrevessem numa prioridade mais prioritária que a de manutenção, seria preciso alterar na interface de testes este valor para ser o mais prioritário. Os testes realizados demonstraram o correto funcionamento da infraestrutura.

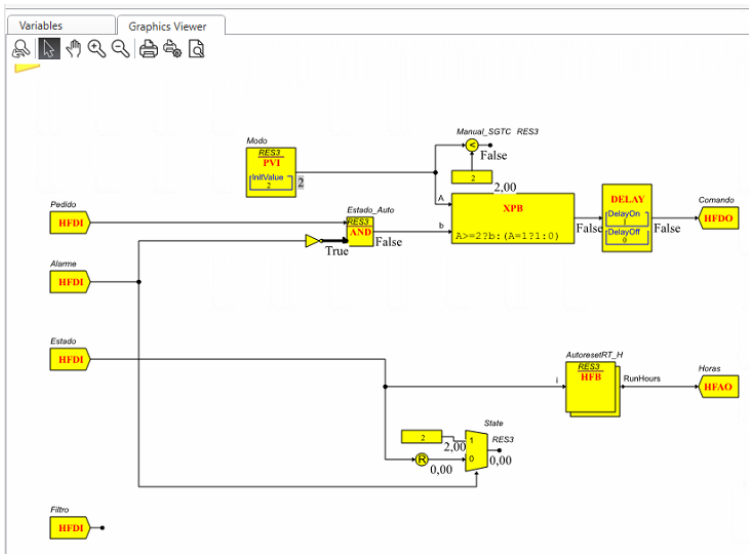


Figura 84. Exemplo de lógica de controle.

Posteriormente, fez-se um Teste Sequencial com o objetivo de atuar num ponto e verificar o resultado noutros pontos. Testaram-se os pontos de estado e alarme de um ventilador. O resultado do teste encontra-se na tabela da figura 85 que foi retirada do relatório criado pela ferramenta no fim do teste.

Equipamento	Ponto	Tipo	BACnet ID	Ação	Resultado
BN_QEAVAC3	VE28_Estado	binaryInput	103	Leitura	Inicial: active Final: inactive Esperado: inactive Resultado: Aprovado
BN_QEAVAC3	VE28_Alarme	binaryInput	104	Leitura	Inicial: inactive Final: active Esperado: active Resultado: Aprovado
BN_QEAVAC3	VE28_Comando	binaryOutput	109	Escrita	Escrito: inactive Prioridade: 8

Figura 85. Relatório de um teste sequencial a um ventilador.

Após a conclusão dos testes realizados, verificou-se que a ferramenta de testes desenvolvida cumpriu de forma satisfatória os objetivos definidos inicialmente. As diferentes funcionalidades implementadas apresentaram o comportamento esperado, permitindo executar os testes nos pontos *BACnet*, registrar resultados e criar relatórios a partir desses testes.

Durante a fase de validação foram identificados alguns problemas e inconsistências. No entanto, concluiu-se que estas situações não estavam diretamente relacionadas com a ferramenta de testes desenvolvida, mas sim com a ferramenta anteriormente criada para criar os ficheiros XML a partir do ficheiro *Excel* para posterior importação no *EBO*.

Desta forma, os testes efetuados permitiram não só validar o funcionamento da ferramenta de testes, mas também identificar falhas existentes na primeira ferramenta desenvolvida, demonstrando a utilidade prática da solução desenvolvida no processo de verificação e validação das instalações através do protocolo *BACnet*. Para além disto, é destacar a redução do tempo necessário para realizar estas atividades. Verificou-se que um trabalho de manutenção que costuma demorar aproximadamente quarenta horas (uma semana de trabalho) pode ser, através desta ferramenta, realizado em menos de oito horas (um dia de trabalho).

5. Conclusões

A realização deste estágio permitiu desenvolver um conjunto de ferramentas destinadas a otimizar o processo de comissionamento de Sistemas de Gestão de Edifícios. O principal objetivo consistiu na redução do tempo necessário para a execução destas tarefas e na diminuição da probabilidade de erro humano através da automatização de processos que, até então, eram realizados manualmente.

A substituição de métodos repetitivos e dependentes de verificação manual por processos digitais automatizados contribuiu para aumentar a eficiência dos trabalhos realizados. O problema inicialmente identificado, relacionado com o tempo elevado necessário para efetuar verificações nas instalações de grande dimensão e com a facilidade de ocorrência de erros humanos, foi abordado através do desenvolvimento de três ferramentas feitas com recurso à linguagem *Python* e diversas bibliotecas auxiliares.

Verificou-se que a ferramenta de testes conseguiu reduzir o tempo necessário para efetuar uma manutenção e que as soluções desenvolvidas demonstraram uma melhoria significativa na metodologia de trabalho. Tradicionalmente, o comissionamento de Sistemas de Gestão de Edifícios envolve a verificação manual de centenas de pontos físicos, através de observação direta e registo manual, o que constitui um processo moroso e suscetível a inconsistências.

A primeira ferramenta, Importação para o *BMS*, permitiu automatizar parte da fase de planeamento e implementação no *software EcoStruxure Building Operation (EBO)*. A criação de forma automática dos ficheiros *XML* e *TGML* que contêm a informação necessária para caracterizar o sistema, assegurou uma maior uniformização das configurações, reduziu erros associados à criação manual de pontos físicos e pontos *BACnet* e conferiu a capacidade de tornar o processo mais rápido.

A segunda ferramenta, Comparador de Versões, permite a análise de alterações entre diferentes versões de um projeto. A possibilidade de identificar rapidamente diferenças em nomes, descrições e identificadores *BACnet* permite facilitar o controlo das alterações efetuadas ao longo do desenvolvimento e manutenção das instalações.

Relativamente à terceira ferramenta, Ferramenta de Testes, a utilização da biblioteca *BAC0* permitiu estabelecer uma comunicação direta com os controladores em tempo real. A implementação dos diferentes testes desenvolvidos permite simular diferentes cenários de funcionamento a fim de validar o bom funcionamento dos diferentes sistemas de uma instalação. O maior impacto foi verificado no Teste de Limites, onde a validação é executada de forma automática, permitindo testar múltiplas entradas analógicas em poucos segundos, algo inviável de realizar manualmente com o mesmo rigor. Nos testes que ainda exigem intervenção humana, como saídas ou testes sequenciais, a ferramenta contribui sobretudo para a centralização da informação, reduzindo a fragmentação entre interfaces e documentação.

Do ponto de vista do desenvolvimento de *software*, um dos desafios esteve relacionado com a adaptação da aplicação para ambiente *Linux*, nomeadamente *Raspberry Pi OS*. Esta adaptação exigiu alterações na gestão de ficheiros *Excel* e ajustes na interface gráfica desenvolvida com a biblioteca *CustomTkinter*. Como

resultado, passou a ser possível utilizar a ferramenta diretamente num equipamento portátil de baixo consumo, permitindo facilitar futuras intervenções de manutenção e diagnóstico de forma remota.

A validação das ferramentas em ambiente real constituiu um dos aspetos mais relevantes deste trabalho. Os testes realizados nos hotéis *The Views Oásis*, Alto Lido e Baía Azul permitiram identificar problemas de configuração que não tinham sido detetados durante a fase de implementação. Entre os problemas encontrados destacam-se a ausência de unidades em determinados pontos *BACnet*, o que impedia a correta leitura de valores analógicos, bem como erros nos *bindings* de pontos de saída, em que a direção das referências estava invertida. Estes problemas não estavam na ferramenta usada para os testes, mas sim na primeira ferramenta desenvolvida.

As ferramentas analisadas no Estado da Arte demonstram que existem soluções comerciais capazes de apoiar diferentes fases do comissionamento de sistemas BMS. No entanto, verificou-se que estas ferramentas apresentam limitações quando consideradas no contexto do trabalho desenvolvido. O *WebCTRL* integra funcionalidades de gestão e apoio ao comissionamento, mas está fortemente associado ao ecossistema da *Automated Logic*. O *EcoStruxure Building Commission* fornece funcionalidades de apoio ao comissionamento para instalações *Schneider Electric*, mas não automatiza a criação da estrutura completa do projeto nem disponibiliza mecanismos de comparação automática entre revisões de listas de pontos. O *AutoCXI* disponibiliza funcionalidades avançadas para testes funcionais, porém constitui uma solução comercial independente e não contempla a automatização das tarefas específicas realizadas durante a implementação de um BMS. A solução desenvolvida neste trabalho reúne três funcionalidades complementares: automatização da criação dos ficheiros de configuração do BMS, ; comparação automática entre versões das listas de pontos; e execução automatizada de testes funcionais com a criação de relatórios. Estas funcionalidades foram concebidas de acordo com as necessidades identificadas na empresa onde decorreu o estágio, permitindo reduzir o tempo para realizar determinadas tarefas do comissionamento, minimizar erros humanos e aumentar a repetibilidade do processo de comissionamento.

De forma geral, o trabalho desenvolvido permitiu introduzir melhorias no processo de comissionamento da empresa de estágio, sobretudo ao nível da automatização de testes e da criação automática de relatórios em formato *PDF*. Para além de facilitar a documentação dos testes realizados, estes relatórios constituem também um registo técnico do funcionamento e validação do sistema.

Conclui-se, assim, que os objetivos inicialmente definidos foram atingidos. As ferramentas desenvolvidas contribuíram para reduzir o tempo necessário para a execução das manutenções nas instalações e diminuir a probabilidade de falhas relacionadas com a configuração inicial do sistema no BMS.

5.1. Trabalhos futuros

Apesar do cumprimento dos objetivos inicialmente propostos e dos resultados positivos obtidos na validação prática, foram identificadas diversas oportunidades de melhoria e expansão que poderão ser exploradas em desenvolvimentos futuros, com o objetivo de aumentar a robustez e a versatilidade do *software* concebido.

Em primeiro lugar, revela-se pertinente a implementação de melhorias no formato utilizado para o armazenamento das amostras recolhidas durante os períodos de monitorização. Atualmente, o sistema de registo em ficheiros CSV apresenta uma estrutura que repete informação técnica de forma redundante em cada entrada, o que poderá ser otimizado através de uma normalização da estrutura de dados. Esta alteração permitiria não só reduzir o espaço ocupado em disco, como também facilitar o processamento e a análise estatística de grandes volumes de dados.

Uma evolução significativa para a ferramenta de testes consistiria na transição de um modelo dependente de importação para um modelo de descoberta dinâmica na rede. No estado atual, a seleção de equipamentos e pontos para teste baseia-se estritamente na leitura de ficheiros *Excel* previamente preparados. A implementação de uma funcionalidade de procura ativa permitiria ao técnico explorar a rede em tempo real e selecionar os equipamentos e objetos de interesse diretamente na interface gráfica, permitindo uma maior autonomia no processo de comissionamento.

Propõe-se também a unificação dos relatórios criados pela aplicação. Atualmente, a ferramenta cria um relatório independente após a conclusão de cada teste individual, o que pode resultar numa fragmentação excessiva da documentação. Sugere-se, portanto, a alteração da lógica de armazenamento dos resultados para permitir a exportação de um relatório final consolidado, que agregue todos os testes realizados numa determinada intervenção técnica. Esta abordagem exigiria uma reestruturação na forma como os dados são mantidos em memória durante a execução do programa, mas garantiria que a documentação fosse mais profissional e organizada para posterior entrega ao cliente final.

Sugere-se uma maior integração entre as ferramentas desenvolvidas, permitindo a reutilização da informação produzida em cada etapa. Desta forma, os dados gerados pela ferramenta de importação poderiam ser utilizados diretamente pela ferramenta de comparação e, posteriormente, pela ferramenta de testes. Esta integração permitiria eliminar tarefas intermédias, reduzir a duplicação de informação e facilitar a utilização das ferramentas.

Por fim, e para acompanhar as tendências de segurança no setor dos Sistemas de Gestão de Edifícios, revela-se fundamental a adaptação do *software* de testes para funcionar com o protocolo *BACnet Secure Connect (BACnet/SC)*. Uma vez que este novo padrão elimina a dependência de mensagens de *broadcast* e utiliza comunicações encriptadas através de *TLS*.

Referências

- [1] Manta, Agência de Comunicação, “BLIN Control - Engenharia, Lda.,” [Online]. Available: <https://blin.pt/>. [Acedido em 11 de junho de 2026].
- [2] A. Devasia, “Building Management Systems: Comprehensive Guide (2024),” Safe and Sound Security, 9 de setembro de 2024. [Online]. Available: <https://getsafeandsound.com/blog/building-management-system/>. [Acedido em 16 de setembro de 2025].
- [3] V. Graveto, T. Cruz e P. Simões, “Security of Building Automation and Control Systems: Survey and future research directions,” 29 de outubro de 2021.
- [4] ISO/TC 205, “ISO 16484-1:2024 Building automation and control systems (BACS),” [Online]. Available: <https://www.iso.org/obp/ui/en/#iso:std:iso:16484:-1:ed-2:v1:en>. [Acedido em 25 de outubro de 2025].
- [5] G. Li, L. Ren, Y. Fu, Z. Yang, V. Adetola, J. Wen, Q. Zhu, T. Wu, K. Candan e Z. O'Neill, “A critical review of cyber-physical security for building automation systems,” 09 de março de 2023.
- [6] P. Zito, A Managers Guide to Building Automation Systems (All the things I wish I knew when I started my career in building automation), Smart Building Academy, 2016.
- [7] “QA Graphics,” [Online]. Available: <https://www.qagraphics.com/>.
- [8] EBC - Energy in Building And Communities, “Anexx 40 - Commissioning of Building HVAC Systems for Improving Energy Performance,” 2001 - 2004. [Online]. Available: http://www.ecbcs.org/Data/publications/EBC_Annex_40_Commissioning_Tools_for_Improved_Energy_Performance.pdf. [Acedido em 25 de setembro de 2025].
- [9] Whole Building Design Guide, “Building Commissioning: The Process,” [Online]. Available: <https://www.wbdg.org/building-commissioning>. [Acedido em 26 de setembro de 2025].
- [10] ASHRAE, “ASHRAE Standards and Guidelines,” [Online]. Available: <https://www.ashrae.org/technical-resources/standards-and-guidelines/read-only-versions-of-ashrae-standards>. [Acedido em 27 de setembro de 2025].
- [11] ASHRAE Bookstore, “Guideline 0-2019 -- The Commissioning Process,” [Online]. Available: https://store accuristech.com/standards/guideline-0-2019-the-commissioning-process?product_id=2076120. [Acedido em 27 de setembro de 2025].
- [12] BSRIA, “Commissioning guides,” [Online]. Available: https://www.bsria.com/uk/knowledge/bookshop/commissioning_guides/. [Acedido em 27 de setembro de 2025].

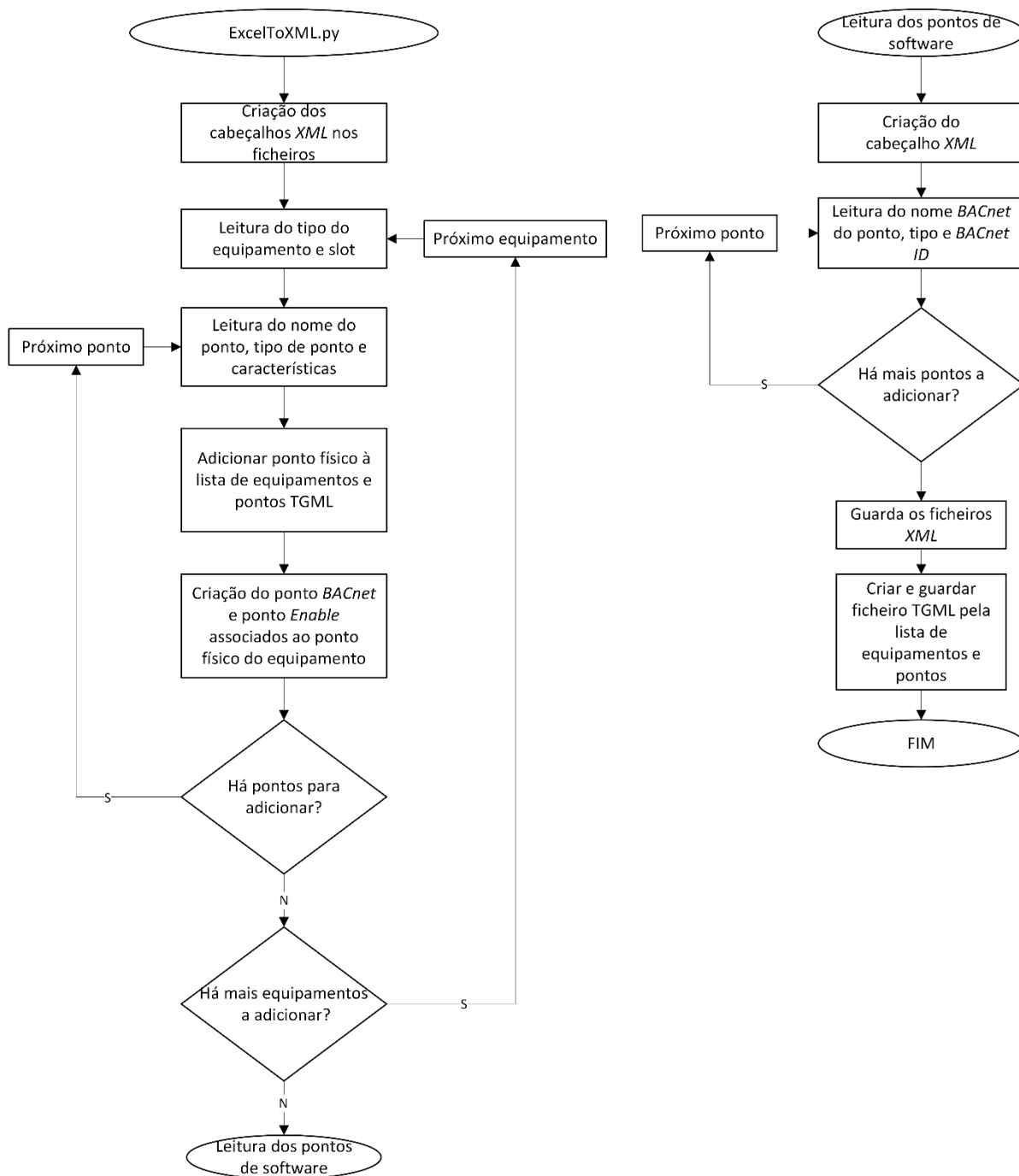
- [13] CIBSE, “CIBSE- Knowledge Portal,” [Online]. Available: <https://www.cibse.org/knowledge-research/knowledge-portal/>.
- [14] Diário da República, “Decreto-Lei n.º 101-D/2020, de 7 de dezembro,” [Online]. Available: <https://diariodarepublica.pt/dr/detalhe/decreto-lei/101-d-2020-150570704>.
- [15] J. Sinopoli, Smart Building Systems for Architects, Owners, and Builders, Butterworth-Heinemann, 2010.
- [16] H. L. K., “The open protocol standard for computerized building systems: BACnet,” Agosto 1999. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/801208>. [Acedido em 7 de maio de 2026].
- [17] M. Nast, B. Butzin, F. Golatowski e D. Timmermann, “Performance Analysis of a Secured BACnet/IP Network,” 2019. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8758009>. [Acedido em 06 de Maio de 2026].
- [18] M. Peacock, M. Johnstone e C. Valli, “Security Issues with BACnet Value Handling,” 2017. [Online]. Available: <https://www.scitepress.org/Papers/2017/62634/62634.pdf>. [Acedido em 11 de maio de 2026].
- [19] D. Fisher, B. Isler e M. Osborne, “BACnet Secure Connect - A Secure Infrastructure for Building Automation,” [Online]. Available: https://www.ashrae.org/file%20library/technical%20resources/bookstore/bacnet-sc-whitepaper-v15_final_20190521.pdf. [Acedido em 23 de setembro de 2025].
- [20] T. Hansemann, C. Hübner e H. Merz, Building Automation Communication Systems with EIB/KNX, LON and BACnet, Segunda ed., Springer, 2018.
- [21] “M-bus Protocol,” Stackforce - Embedded Connectivity Solutions, [Online]. Available: <https://stackforce.com/en/portfolio/technologies/m-bus>. [Acedido em 22 de setembro de 2025].
- [22] “Technical Support What is DALI,” NVC Lighting, [Online]. Available: <https://www.nvcuk.com/technical-support/view/what-is-dali-8>. [Acedido em 22 setembro de 2025].
- [23] “Como funciona a iluminação DALI? – Um guia abrangente,” Logoslighting, [Online]. Available: <https://logoslighting.com/pt/blog/how-does-dali-lighting-work-a-comprehensive-guide/>. [Acedido em 21 de setembro de 2025].
- [24] Automated Logic, “ACxelerate™ Commissioning Tool,” [Online]. Available: <https://www.automatedlogic.com/en/products/webctrl-building-automation-system/engineering-tools/acxelerate/>. [Acedido em 30 de setembro de 2025].
- [25] Schneider Electric, “EcoStruxure Building Commission,” [Online]. Available: <https://ecostruxurebuildinghelp.se.com/bms/Topics/Show.castle?id=13861&locale=en-US&productversion=2.0>. [Acedido em 30 de setembro de 2025].

- [26] Brandon Systems, "AutoCXI - BMS Automated Continuous Commissioning Systems," [Online]. Available: <https://autocxi.com/>. [Acedido em 30 de setembro de 2025].
- [27] M. Jradi, . N. Boel, B. E. Madsen, . J. Jacobsen, J. S. Hooge e L. Kildelund, "BuildCOM: automated auditing and continuous commissioning of next generation building management systems," 13 de janeiro de 2021. [Online]. Available: <https://link.springer.com/article/10.1186/s42162-020-00136-2>. [Acedido em 02 de janeiro de 2026].
- [28] J. Bender, "BACpypes Documentation," BACpypes, [Online]. Available: <https://bacpypes.readthedocs.io/>. [Acedido em 15 de novembro de 2025].
- [29] OpenJS Foundation and Node-RED contributors, "Low-code programming for event-driven applications," [Online]. Available: <https://nodered.org/>. [Acedido em 15 de novembro de 2025].
- [30] Contemporary Controls, "BACnet® Discovery Tool (BDT) Instructions," [Online]. Available: <https://www.ccontrols.com/pdf/is/IS-BDTv214.pdf>. [Acedido em 16 de novembro de 2025].
- [31] "BAC0 Documentation," [Online]. Available: <https://bac0.readthedocs.io/en/latest/index.html>. [Acedido em 16 de novembro de 2025].
- [32] A. Smagulova e A. E. Cerpa, "FADED: FAUlt DETection and Diagnostic System for HVAC Sensors in Commercial Buildings," [Online]. Available: <https://ewsn.org/file-repository/ewsn2024/ewsn24-final89.pdf>. [Acedido em 19 novembro de 25].
- [33] J. J. M. Jimenez, L. Stokes, Q. Yang e V. N. Livina, "Early warning signals of failures in building management systems," [Online]. Available: https://www.researchgate.net/publication/351874915_Early_warning_signals_of_failures_in_building_management_systems#fullTextFileContent. [Acedido em 21 novembro de 2025].
- [34] W. Nelson e C. Dieckert, "Machine Learning-Based Automated Fault Detection and Diagnostics in Building Systems," [Online]. Available: <https://www.mdpi.com/1996-1073/17/2/529>. [Acedido em 21 novembro de 2025].
- [35] J. Morewood, "Building energy performance monitoring through the lens of data quality: A review," 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378778822008726>. [Acedido em 26 de novembro de 2025].
- [36] "British Standards Institution, Data quality, Standards, Vol. BSI ISO 8000, British Standards Institution, London, United Kingdom, 2022," [Online]. Available: <https://www.bsigroup.com/>. [Acedido em 26 de novembro de 2025].
- [37] "ISO 8000-110:2021(en) Data quality — Part 110: Master data: Exchange of characteristic data: Syntax, semantic encoding, and conformance to data specification," [Online]. Available:

<https://www.iso.org/obp/ui/en/#iso:std:iso:8000:-110:ed-2:v1:en>. [Acedido em 26 de novembro de 2025].

- [38] “BACpypes3 - Documentação,” [Online]. Available: <https://bacpypes3.readthedocs.io/en/latest/gettingstarted/setup.html>. [Acedido em 19 de janeiro de 2025].
- [39] “pyBACnet,” [Online]. Available: <https://github.com/hoangdung2/pyBACnet>. [Acedido em 20 de janeiro de 2026].
- [40] “BACnetPY,” [Online]. Available: <https://github.com/chr1s-t0pher/BACnetPY>. [Acedido em 20 de janeiro de 2026].
- [41] R. M. L. “JOY-IT RB-CaseP4-07B - Caixa de calha DIN para Raspberry Pi 4 - Preta,” [Online]. Available: <https://mauser.pt/095-7844/joy-it-rb-casep4-07b-caixa-de-calha-din-para-raspberry-pi-4-preta>. [Acedido em 15 de maio de 2026].

Anexo A – Fluxograma da ferramenta de criação dos ficheiros *XML*



Anexo B – Dicionários de campos extra

Anexo B.1. – Pontos Físicos (Entradas/Saídas)

```
EXTRA_FIELDS = {  
    "io.point.CurrentInput": [  
        {"Name": "ElecBottomOfScale", "col": COL_ELEC_BOTTOM_SCALE},  
        {"Name": "ElecTopOfScale", "col": COL_ELEC_TOP_SCALE},  
        {"Name": "EngBottomOfScale", "col": COL_ENG_BOTTOM_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "EngTopOfScale", "col": COL_ENG_TOP_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "LogArray", "unit_col": COL_ENG_UNIT},  
        {"Name": "Offset", "unit_col": COL_ENG_UNIT, "Value": "0"},  
        {"Name": "Value", "unit_col": COL_ENG_UNIT},  
    ],  
    "io.point.VoltageInput": [  
        {"Name": "ElecBottomOfScale", "col": COL_ELEC_BOTTOM_SCALE},  
        {"Name": "ElecTopOfScale", "col": COL_ELEC_TOP_SCALE},  
        {"Name": "EngBottomOfScale", "col": COL_ENG_BOTTOM_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "EngTopOfScale", "col": COL_ENG_TOP_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "LogArray", "unit_col": COL_ENG_UNIT},  
        {"Name": "Offset", "unit_col": COL_ENG_UNIT, "Value": "0"},  
        {"Name": "Value", "unit_col": COL_ENG_UNIT},  
    ],  
    "io.point.VoltageOutput": [  
        {"Name": "ElecBottomOfScale", "col": COL_ELEC_BOTTOM_SCALE},  
        {"Name": "ElecTopOfScale", "col": COL_ELEC_TOP_SCALE},  
        {"Name": "EngBottomOfScale", "col": COL_ENG_BOTTOM_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "EngTopOfScale", "col": COL_ENG_TOP_SCALE, "unit_col": COL_ENG_UNIT},  
        {"Name": "LogArray", "unit_col": COL_ENG_UNIT},  
        #{"Name": "Offset", "unit_col": COL_ENG_UNIT, "Value": "0"},  
        {"Name": "Value", "unit_col": COL_ENG_UNIT},  
    ],  
    "io.point.CounterInput": [  
        {"Name": "LEDColor", "col": COL_LED_COLOR, "value_map": "LED_COLOR"}  
    ],  
    "io.point.DigitalOutput": [  
        {"Name": "RequestedValue", "Value": "0"}  
    ],  
    "io.point.DigitalInput": [  

```

```

    {"Name": "LEDColor", "col": COL_LED_COLOR, "value_map": "LED_COLOR"}
],
"io.point.ResistiveInput": [
    {"Name": "ElecBottomOfScale", "col": COL_ELEC_BOTTOM_SCALE},
    {"Name": "ElecTopOfScale", "col": COL_ELEC_TOP_SCALE},
    {"Name": "EngBottomOfScale", "col": COL_ENG_BOTTOM_SCALE, "unit_col": COL_ENG_UNIT},
    {"Name": "EngTopOfScale", "col": COL_ENG_TOP_SCALE, "unit_col": COL_ENG_UNIT},
    {"Name": "LogArray", "unit_col": COL_ENG_UNIT},
    {"Name": "Value", "unit_col": COL_ENG_UNIT},
],
"io.point.TemperatureInput": [
    {"Name": "ThermistorType", "col": COL_SENSOR_TEMP, "value_map": "SENSOR_TEMP"},
],
"io.point.Exemplos": [
    # Name = Exemplo1 Value = valor da coluna
    {"Name": "Exemplo1", "col": COL_ELEC_BOTTOM_SCALE},
    # Name="Exemplo2" Value="valorDaColuna" Unit="valorDaColuna"
    {"Name": "Exemplo2", "col": COL_ELEC_BOTTOM_SCALE, "unit_col": COL_ENG_UNIT},
    # Name="Exemplo3" Value="valorDaColuna" Unit="0x310001"
    {"Name": "Exemplo3", "col": COL_ELEC_BOTTOM_SCALE, "unit_value": "0x310001"},
    # Name="Exemplo4" Value="mapeamento do valor da coluna 11 no dicionário Sensor_Temp"
    {"Name": "Exemplo4", "col": 11, "value_map": "SENSOR_TEMP"},
],
}

```

Anexo B.2. – Pontos BACnet

```

BACNET_EXTRA_FIELDS = {
    "bacnet.point.digital.Input": [
        {"Name": "AlarmValue", "Value": "1"},
        {"Name": "AssociatedValue"},
        {"Name": "BACnetName"},
        #{"Name": "Value"},
        # Campos comuns (presentes se houver valor no Excel)
        {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},
        {"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
        {"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
        {"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
        {"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
    ]
}

```

```

# Campos específicos de digital input
#Não tem
],
"bacnet.point.digital.Output": [
  {"Name": "BACnetName"},
  {"Name": "RelinquishDefault", "Value": "0"},
  {"Name": "Value"},
  # Campos comuns
  {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},
  {"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
  {"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
  {"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
  {"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
  # Campos específicos de digital output
  {"Name": "MinimumOffTime", "ExcelCol": COL_BACNET_MIN_OFF_TIME},
  {"Name": "MinimumOnTime", "ExcelCol": COL_BACNET_MIN_ON_TIME},
],
"bacnet.point.analog.Input": [
  {"Name": "AssociatedValue"},
  {"Name": "BACnetName"},
  # Campos comuns
  {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},
  {"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
  {"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
  {"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
  {"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
  # Campos específicos de analógico
  {"Name": "HighLimit", "ExcelCol": COL_BACNET_HIGH_LIMIT},
  {"Name": "LowLimit", "ExcelCol": COL_BACNET_LOW_LIMIT},
  {"Name": "MaxPresValue", "ExcelCol": COL_BACNET_MAX_PRES_VALUE},
  {"Name": "MinPresValue", "ExcelCol": COL_BACNET_MIN_PRES_VALUE},
  {"Name": "LimitEnable", "ExcelCol": COL_BACNET_LIMIT_ENABLE},
],
"bacnet.point.analog.Output": [
  {"Name": "BACnetName"},
  {"Name": "Value"},
  # Campos comuns
  {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},

```

```

{"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
{"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
{"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
{"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
# Campos específicos de analógico
{"Name": "HighLimit", "ExcelCol": COL_BACNET_HIGH_LIMIT},
{"Name": "LowLimit", "ExcelCol": COL_BACNET_LOW_LIMIT},
{"Name": "MaxPresValue", "ExcelCol": COL_BACNET_MAX_PRES_VALUE},
{"Name": "MinPresValue", "ExcelCol": COL_BACNET_MIN_PRES_VALUE},
{"Name": "LimitEnable", "ExcelCol": COL_BACNET_LIMIT_ENABLE},
],
"bacnet.point.analog.Value": [
  {"Name": "BACnetName"},
  {"Name": "Value"},
  # Campos comuns
  {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},
  {"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
  {"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
  {"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
  {"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
  # Campos específicos de analógico
  {"Name": "HighLimit", "ExcelCol": COL_BACNET_HIGH_LIMIT},
  {"Name": "LowLimit", "ExcelCol": COL_BACNET_LOW_LIMIT},
  {"Name": "MaxPresValue", "ExcelCol": COL_BACNET_MAX_PRES_VALUE},
  {"Name": "MinPresValue", "ExcelCol": COL_BACNET_MIN_PRES_VALUE},
  {"Name": "LimitEnable", "ExcelCol": COL_BACNET_LIMIT_ENABLE},
],
"bacnet.point.digital.Value": [
  {"Name": "BACnetName"},
  {"Name": "RelinquishDefault", "Value": "0"},
  {"Name": "Value"},
  # Campos comuns
  {"Name": "AlarmMessage", "ExcelCol": COL_BACNET_ALARM_MESSAGE},
  {"Name": "FaultMessage", "ExcelCol": COL_BACNET_FAULT_MESSAGE},
  {"Name": "ResetMessage", "ExcelCol": COL_BACNET_RESET_MESSAGE},
  {"Name": "ResetDelay", "ExcelCol": COL_BACNET_RESET_DELAY},
  {"Name": "TimeDelay", "ExcelCol": COL_BACNET_TIME_DELAY},
  # Campos específicos de digital output

```

```
    {"Name": "MinimumOffTime", "ExcelCol": COL_BACNET_MIN_OFF_TIME},  
    {"Name": "MinimumOnTime", "ExcelCol": COL_BACNET_MIN_ON_TIME},  
],  
}
```

Anexo C – Código de comparação de listas de pontos

```
def comparar_listas_pontos(lista1, lista2, tipo):
    dict1 = indexar_pontos_por_row(lista1)
    dict2 = indexar_pontos_por_row(lista2)
    rows1 = set(dict1.keys())
    rows2 = set(dict2.keys())
    resultado = []
    # Pontos só na lista 1
    for row in rows1 - rows2:
        p1 = dict1[row]
        rp = ResultadoPonto(p1.nome, p1.tipo, p1.index, row, tipo)
        rp.status = "erro"
        rp.mensagens.append("Existe apenas no ficheiro A")
        resultado.append(rp)
    # Pontos só na lista 2
    for row in rows2 - rows1:
        p2 = dict2[row]
        rp = ResultadoPonto(p2.nome, p2.tipo, p2.index, row, tipo)
        rp.status = "erro"
        rp.mensagens.append("Existe apenas no ficheiro B")
        resultado.append(rp)
    # Pontos na mesma row → comparar propriedades
    for row in rows1 & rows2:
        p1 = dict1[row]
        p2 = dict2[row]
        rp = ResultadoPonto(p1.nome, p1.tipo, p1.index, row, tipo)
        alteracoes = []
        if (p1.nome or "").strip() != (p2.nome or "").strip():
            alteracoes.append(f"Nome: '{p1.nome}' → '{p2.nome}'")
        if p1.tipo != p2.tipo:
            alteracoes.append(f"Tipo: '{p1.tipo}' → '{p2.tipo}'")
        if (p1.descricao or "").strip() != (p2.descricao or "").strip():
            alteracoes.append(f"Descrição alterada")
        if str(p1.index) != str(p2.index):
```

```
        alteracoes.append(f"ID: {p1.index} → {p2.index}")
    if alteracoes:
        rp.status = "alterado"
        rp.mensagens.extend(alteracoes)
    resultado.append(rp)
return resultado
```

Anexo D – Código de procura de equipamentos e leitura de parâmetros *BACnet*

```
import asyncio
```

```
import BAC0
```

```
async def main():
```

```
    async with BAC0.start() as bacnet:
```

```
        # Descoberta global
```

```
        await bacnet._discover(global_broadcast=True)
```

```
        # Esperar dinamicamente até que a descoberta termine
```

```
        timeout = 5 # segundos sem novos dispositivos para encerrar
```

```
        last_count = 0
```

```
        last_change_time = asyncio.get_event_loop().time()
```

```
        while True:
```

```
            current_count = len(bacnet.discoveredDevices)
```

```
            if current_count != last_count:
```

```
                last_count = current_count
```

```
                last_change_time = asyncio.get_event_loop().time()
```

```
            elif asyncio.get_event_loop().time() - last_change_time > timeout:
```

```
                break # nenhum novo dispositivo chegou no timeout
```

```
            await asyncio.sleep(0.1) # pequena pausa para não travar o loop
```

```
        #Dispositivos encontrados estão em bacnet.discoveredDevices
```

```
        print("Dispositivos encontrados:")
```

```
        print(bacnet.discoveredDevices)
```

```
        target_ip = "192.168.1.241"
```

```
        for key, info in bacnet.discoveredDevices.items():
```

```
            if str(info["address"]) == target_ip:
```

```
                device_id = info["object_instance"][1]
```

```
                print(f"Dispositivo encontrado: ID {device_id} no IP {target_ip}")
```

```
                print(f"Chaves disponíveis para {key}: {list(info.keys())}")
```

```
                """
```

```
                {
```

```
                'object_instance': (<ObjectType: device>, 100527),
```

```
                'address': <IPv4Address 192.168.1.241>,
```

```

'network_number': {1},
'vendor_id': 10,
'vendor_name': 'unknown'
}
"""

all_points = await bacnet.readMultiple(f"{target_ip} device {device_id} all")
#print(all_points)
for i, item in enumerate(all_points):
    print(f"[{i}] ({type(item).__name__}): {item}")
print("\n\n")
ponto = await bacnet.read(f"{target_ip} analogInput 18 presentValue")
unidades = await bacnet.read(f"{target_ip} analogInput 18 units")
nome = await bacnet.read(f"{target_ip} analogInput 18 objectName")
descricao = await bacnet.read(f"{target_ip} analogInput 18 description")
print(f"AnalogInput [18] {ponto}, {unidades},{nome}, {descricao}")

"""

ponto = await bacnet.read(f"{target_ip} binaryInput 31 presentValue")
status = await bacnet.read(f"{target_ip} binaryInput 31 statusFlags")
desc = await bacnet.read(f"{target_ip} binaryInput 31 description")
print(f"BinaryInput[31] {ponto}, {status}, {desc}")

ponto = await bacnet.read(f"{target_ip} binaryValue 32 presentValue")
status = await bacnet.read(f"{target_ip} binaryValue 32 statusFlags")
desc = await bacnet.read(f"{target_ip} binaryValue 32 description")
print(f"BinaryValue[32] {ponto}, {status}, {desc}")

"""

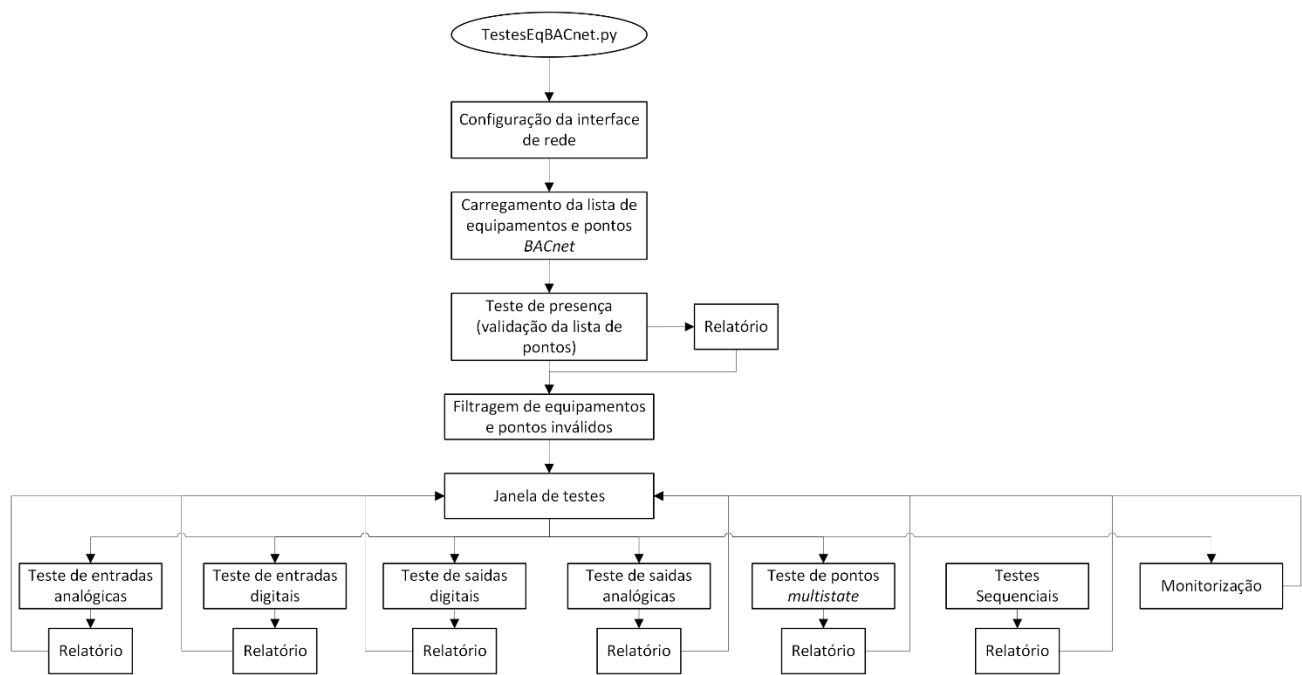
# Escrever valor 0 com prioridade 8
await bacnet._write(f"{target_ip} binaryValue 32 presentValue 0 - 8")
asyncio.run(main())

```

Anexo E – Tabela de filtros para o Teste de Limites

Categoria	Subcategoria	Limite Inferior	Limite Superior	Unidade (codificada)	Unidade
Temperaturas	Congelados	-25	0	degrees-celsius	°C
	Refrigerados	0	12	degrees-celsius	°C
	Água Gelada	3	15	degrees-celsius	°C
	Ambiente	10	40	degrees-celsius	°C
	Bombas de Calor	25	70	degrees-celsius	°C
	Água Fria	10	25	degrees-celsius	°C
	AQS	25	60	degrees-celsius	°C
	Água Quente	25	95	degrees-celsius	°C
	Solar	10	125	degrees-celsius	°C
	Vapor	50	150	degrees-celsius	°C
Caudal	Evaporador	5	250	cubic-meters-per-hour	m3/h
	Condensador	10	300	cubic-meters-per-hour	m3/h
	Água	0	50	liters-per-second	L/s
	Ar	0	25000	cubic-meters-per-hour	m3/h
Pressão	Sucção	-10	300	pascals	Pa
	Descarga	0	500	pascals	Pa
	Relativa Água	0	10	bars	bar
	Relativa Ar	0	2500	pascals	Pa
	Diferencial Água	0	6	bars	bar
	Diferencial Ar	0	500	pascals	Pa
Resistência		0	10000	ohms	Ohm
Capacitancia		0	1000	farads	F
Capacidade		0	120	percent	%
Frequência	Rede	49	51	hertz	Hz
	Variador	0	100	hertz	Hz
Velocidade	rpm	0	3000	revolutions-per-minute	rpm
	Ar	0	30	meters-per-second	m/s
Abertura	Válvula	0	100	percent	%
	Registo	0	100	percent	%
Corrente	Nominal	0	3000	amperes	A
Tensão	Simples	215	245	volts	V
	Composta	380	420	volts	V
Humidade	Relativa	25	95	percent	%
pH		5,5	8	no-units	
Cloro		0	3	no-units	
Bromo		0	3	no-units	
CO2		400	700	parts-per-million	ppm
VOC		15	50	percent	%
Luminosidade	Interior	0	400	luxes	Lux
	Exterior	0	15000	luxes	Lux

Anexo F – Fluxograma do funcionamento da ferramenta de testes



Anexo G – Carregamento da lista de pontos na ferramenta de testes

Validação BACnet

Ficheiro Excel: C:\Users\abreu\OneDrive - Universidade da Madeira\5ºAno\Tese\ControloBACnet\TestesEqBACnet.xlsx

Equipamentos → 51 0 0 Pontos → 147 0 0 Timeout BACnet (s) 5

Procurar

Carregar Excel

Validar Rede BACnet

Guardar Excel Atualizado

Mostrar apenas erros e alertas

▼ Equipamento: RP-C-16A_TesteBACnet [IP=None, BACnetID=None]

— Ponto: Resistor Input [Tipo=analogInput, BACnetID=None]

— Ponto: Analog Value [Tipo=analogValue, BACnetID=None]

— Ponto: Digital Input [Tipo=booleanInput, BACnetID=55]

— Ponto: C [Tipo=None, BACnetID=None]

— Ponto: Digital Output [Tipo=None, BACnetID=None]

— Ponto: ID 59 [Tipo=None, BACnetID=59]

— Ponto: ID 60 [Tipo=None, BACnetID=60]

— Ponto: ID 61 [Tipo=None, BACnetID=61]

— Ponto: ID 62 [Tipo=None, BACnetID=62]

— Ponto: ID 63 [Tipo=None, BACnetID=63]

— Ponto: ID 64 [Tipo=None, BACnetID=64]

— Ponto: ID 65 [Tipo=None, BACnetID=65]

— Ponto: ID 66 [Tipo=None, BACnetID=66]

— Ponto: ID 67 [Tipo=None, BACnetID=67]

— Ponto: ID 68 [Tipo=None, BACnetID=68]

— Ponto: ID 69 [Tipo=None, BACnetID=69]

— Ponto: ID 70 [Tipo=None, BACnetID=70]

— Ponto: ID 71 [Tipo=None, BACnetID=71]

— Ponto: ID 72 [Tipo=None, BACnetID=72]

— Ponto: ID 73 [Tipo=None, BACnetID=73]

— Ponto: ID 74 [Tipo=None, BACnetID=74]

— Ponto: ID 75 [Tipo=None, BACnetID=75]

— Ponto: ID 76 [Tipo=None, BACnetID=76]

— Ponto: ID 77 [Tipo=None, BACnetID=77]

— Ponto: ID 78 [Tipo=None, BACnetID=78]

— Ponto: ID 79 [Tipo=None, BACnetID=79]

— Ponto: ID 80 [Tipo=None, BACnetID=80]

— Ponto: ID 81 [Tipo=None, BACnetID=81]

— Ponto: ID 82 [Tipo=None, BACnetID=82]

— Ponto: Digital Value 2 [Tipo=booleanValue, BACnetID=36]

▼ Equipamento: ES_0 [IP=None, BACnetID=None]

— Ponto: f [Tipo=None, BACnetID=None]

— Ponto: g [Tipo=None, BACnetID=None]

— Ponto: ID 55 [Tipo=None, BACnetID=55]

— Ponto: ID 56 [Tipo=None, BACnetID=56]

▼ Equipamento: a [IP=None, BACnetID=None]

— Ponto: Analog Value [Tipo=booleanInput, BACnetID=78]

▼ Equipamento: ES_JN [IP=None, BACnetID=2526]

— Ponto: ID 9101 [Tipo=None, BACnetID=9101]

— Ponto: ID 9102 [Tipo=None, BACnetID=9102]

— Ponto: ID 9103 [Tipo=None, BACnetID=9103]

Anexo H – Página de testes *BACnet*

Página de Testes BACnet

X

Voltar

Equipamento: RaspberryPi-BACnet [IP=192.168.1.5, BACnetID=599]

Ponto: analogInput_1 [Tipo=analogInput, BACnetID=100]

Ponto: analogInput_2 [Tipo=analogInput, BACnetID=101]

Ponto: analogInput_3 [Tipo=analogInput, BACnetID=102]

Ponto: analogInput_4 [Tipo=analogInput, BACnetID=103]

Ponto: analogInput_5 [Tipo=analogInput, BACnetID=104]

Ponto: binaryInput_1 [Tipo=binaryInput, BACnetID=105]

Ponto: binaryInput_2 [Tipo=binaryInput, BACnetID=106]

Ponto: binaryInput_3 [Tipo=binaryInput, BACnetID=107]

Ponto: analogValue_1 [Tipo=analogValue, BACnetID=108]

Ponto: analogValue_2 [Tipo=analogValue, BACnetID=109]

Ponto: analogValue_3 [Tipo=analogValue, BACnetID=110]

Ponto: analogOutput_1 [Tipo=analogOutput, BACnetID=111]

Ponto: analogOutput_2 [Tipo=analogOutput, BACnetID=112]

Ponto: analogOutput_3 [Tipo=analogOutput, BACnetID=113]

Ponto: binaryValue_1 [Tipo=binaryValue, BACnetID=114]

Ponto: binaryValue_2 [Tipo=binaryValue, BACnetID=115]

Ponto: binaryValue_3 [Tipo=binaryValue, BACnetID=116]

Ponto: binaryOutput_1 [Tipo=binaryOutput, BACnetID=117]

Ponto: binaryOutput_2 [Tipo=binaryOutput, BACnetID=118]

Ponto: binaryOutput_3 [Tipo=binaryOutput, BACnetID=119]

Guardar Relatório

Editar mensagens

Mostrar apenas erros/aviso

Límites Entradas Analógicas

Teste Entradas Digitais

Teste Saídas Digitais

Teste Saídas Analógicas

Teste MultiState

Monitorização

Anexo I – Código para a escrita de referências

Anexo I.1. – Nova versão corrigida

```
if bacnet_folder_used:
    Object_path = f"../../../../IO Bus/{slot_card}/{phys_full_name}"
    Object_path_for_IO = f"../../../../Interface
BACnet/Application/{folder_name.strip()}/{bacnet_name}"
else:
    Object_path = f"../../../../IO Bus/{slot_card}/{phys_full_name}"
    Object_path_for_IO = f"../../../../Interface BACnet/Application/{bacnet_name}"
# Reference apenas para inputs/outputs corretos
if bacnet_type.endswith("Input") and pi_def["Name"] == "AssociatedValue":
    ET.SubElement(
        pi, "Reference",
        DeltaFilter="0",
        Object=Object_path,
        Property="Value",
        Retransmit="0",
        TransferRate="10"
    )
elif bacnet_type.endswith("Output") and pi_def["Name"] == "Value":
    """
    ET.SubElement(
        pi, "Reference",
        DeltaFilter="0",
        Object=Object_path,
        Property="Value", #RequestedValue
        Retransmit="0",
        TransferRate="10"
    )
    """
    pi_oi = ET.SubElement(sub_oi, "PI", Name="RequestedValue", Value = "0")
    ET.SubElement( #Referência no ficheiro IO_BUS leitura da variável bacnet
        pi_oi, "Reference",
        DeltaFilter="0",
```

```

        Object=Object_path_for_IO,
        Property="Value", #RequestedValue
        Retransmit="0",
        TransferRate="10"
    )

elif bacnet_type.endswith("Value") and pi_def["Name"] == "Value":
    if "input" in phys_type.lower():
        ET.SubElement(
            pi, "Reference",
            DeltaFilter="0",
            Object=Object_path,
            Property="Value",
            Retransmit="0",
            TransferRate="10"
        )
    elif "output" in phys_type.lower(): #Ex: phys_type = io.point.CurrentInput
        pi_oi = ET.SubElement(sub_oi, "PI", Name="RequestedValue", Value = "0")
        ET.SubElement( #Referência no ficheiro IO_BUS leitura da variavel bacnet
            pi_oi, "Reference",
            DeltaFilter="0",
            Object=Object_path_for_IO,
            Property="Value", #RequestedValue
            Retransmit="0",
            TransferRate="10"
        )

```

Anexo I.2. – Versão antiga

```

if bacnet_folder_used:
    Object_path = f"../../../IO Bus/{slot_card}/{phys_full_name}"
else:
    Object_path = f"../../../IO Bus/{slot_card}/{phys_full_name}"
# Reference apenas para inputs/outputs corretos
if bacnet_type.endswith("Input") and pi_def["Name"] == "AssociatedValue":
    ET.SubElement(

```

```

        pi, "Reference",
        DeltaFilter="0",
        Object=Object_path,
        Property="Value",
        Retransmit="0",
        TransferRate="10"
    )
elif bacnet_type.endswith("Output") and pi_def["Name"] == "Value":
    ET.SubElement(
        pi, "Reference",
        DeltaFilter="0",
        Object=Object_path,
        Property="Value", #RequestedValue
        Retransmit="0",
        TransferRate="10"
    )

elif bacnet_type.endswith("Value") and pi_def["Name"] == "Value":
    ET.SubElement(
        pi, "Reference",
        DeltaFilter="0",
        Object=Object_path,
        Property="Value",
        Retransmit="0",
        TransferRate="10"
    )

```